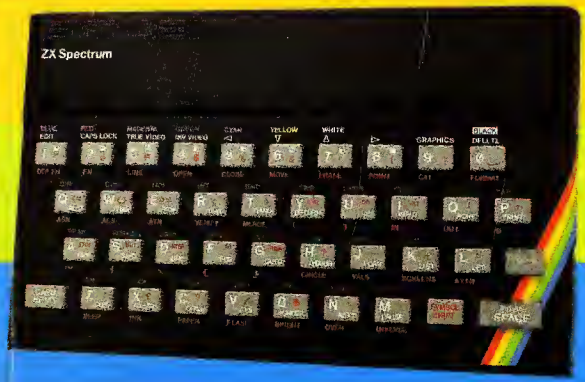
ISBN 90 6215 076 4
D/1983/1997/8
Stewart/Jones, ZX Spectrum

Ian Stewart en Robin Jones

BASIC met de ZX SPECTRUM

BASIC
met de
ZX SPECTRUM

Ian Stewart en
Robin Jones



MAARTEN KLUWER'S
INTERNATIONALE UITGEVERSONDERNEMING
Antwerpen - Apeldoorn

BASIC
met de
ZX SPECTRUM

Ian Stewart – Robin Jones

BASIC

met de

ZX SPECTRUM

MAARTEN KLUWER'S
INTERNATIONALE UITGEVERSONDERNEMING
Antwerpen – Apeldoorn

Oorspronkelijke titel : Easy programming for the ZX Spectrum
© 1982 Shiva Publ. Ltd., Nantwich, Cheshire
Vertaling : L. A. P. van den Wyngaert

BASIC met de ZX Spectrum
ISBN 90 6215 076 4
D/1983/1997/8
UGI Code A 650
Boekhandelsrubriek 16

© 1983 MAARTEN KLUWER'S
INTERNATIONALE UITGEVERSONDERNEMING
Antwerpen – Apeldoorn

Behoudens uitzondering door de Wet gesteld, mag zonder schriftelijke toestemming van de rechthebbende(n) op het auteursrecht, c.q. de uitgever van deze uitgave, door de rechthebbende(n) gemachtigd namens hem (hen) op te treden, niets uit deze uitgave worden verveelvoudigd en/of openbaar gemaakt door middel van druk, fotocopie, microfilm of anderszins, hetgeen ook van toepassing is op de gehele of gedeeltelijke bewerking.
De uitgever is met uitsluiting van ieder ander onherroepelijk door de auteur gemachtigd de door derden verschuldigde vergoeding voor copiëren, als bedoeld in artikel 17 lid 2 der Auteurswet 1912 en in het KB. van 20.6.74 (*Stb.* 351) ex-artikel 16b der Auteurswet 1912, te doen innen door en overeenkomstig de reglementen van de Stichting Reprorecht te Amsterdam.

No part of this book may be reproduced in any form, by print, photoprint, microfilm or any other means, without written permission from the publisher.

Ondanks alle aan de samenstelling van de tekst bestede zorg, kan noch de redactie noch de uitgever noch de vertaler aansprakelijkheid aanvaarden voor eventuele schade, die zou kunnen voortvloeien uit enige fout, die in deze uitgave zou kunnen voorkomen.

INHOUDSOPGAVE

	Voorwoord	7
1	Fundamentele Basic (Hoe communiceren we met de Spectrum)	9
2	Rekenkunde in Basic (+, –, *, /, , LET, INPUT, PRINT)	13
3	Het toetsenbord (shifttoetsen, modes, hoe typt U in wat U wilt)	17
4	Heeeeellllppp!	18
5	Invoer / uitvoer (INPUT "Berichtje", PRINT AT, TAB)	19
6	Lussen (FOR, NEXT)	21
7	Debugging I (Errortypes)	24
8	Toevalsgetallen (Random Numbers RND)	28
9	Sprongen (IF, THEN, GO TO, AND, OR, NOT)	29
10	Plotten (PLOT, DRAW, CIRCLE, OVER, SCREEN$)	32
11	Debugging II (Dry running)	41
12	Grafiek (PRINT AT, INKEY$, PAUSE)	45
13	Door de gebruiker gedefinieerde karakters (POKE, USR, BIN)	50
14	Subroutines (GO SUB, RETURN)	52
15	Klank- en lichtspel (INK, PAPER, BORDER, BEEP, FLASH)	56
16	Debugging III (Tracing)	63
17	Strings ($, CODE, VAL, LEN, TO)	69
18	Gegevens (DATA, READ, RESTORE)	73
19	Krommen uitplotten (Voor de wiskundigen)	75
20	Debugging IV (Afrondfouten)	80
21	Programmeerstijl (en een rakettenafweer-spelletje)	81
22	Peek en poke (PEEK, POKE)	87
23	Tips (Kneepjes en truukjes)	91
24	Kant en klaar - programma's (Copieer en RUN ze)	94

VOORWOORD

Na een half miljoen ZX 81 microcomputers te hebben verkocht, komt Clive Sinclair nu met de ZX Spectrum op de markt. Voor een redelijke prijs krijgt U thuis de beschikking over een computer met geluid, kleur en grafieken met een groot oplossend vermogen.(En een acceptabel toetsenbord !)
De Spectrum heeft menig kenmerk gemeen met de ZX 81, daarom behandelen we ze zoveel mogelijk op dezelfde wijze.
De opzet van dit boek is als volgt : dié eigenschappen van de ZX Spectrum op eenvoudige en verstaanbare wijze beschrijven die de beginnende gebruiker van dit systeem onder de knie moet zien te krijgen. Bovendien bevat dit boek ongeveer 30 "kant-en-klare" – programmalistings die U kunt copiëren en runnen wanneer U wilt, en ontmoet U tijdens de lectuur nog een dertigtal andere programma's. Dit boek heeft als voordeel dat we selectief te werk zijn gegaan en ons hebben kunnen concentreren op de minder esoterische kenmerken : doet men dit niet dan riskeert men door de bomen het bos niet meer te zien
Wat hebt U aan dit boek? Het is een complete handleiding voor het programmeren in BASIC, voor het verwerven van een programmeerstijl Daarnaast is het een handleiding om kleur, geluid, bewegende grafieken, kleurgrafieken met een hoog oplossend vermogen, "debugging", "number-crunching" en "string-handling" in Uw programma's toe te passen Als U dit boek hebt doorgewerkt zal het *Handboek* bij de ZX Spectrum gesneden koek voor U zijn
Het geheel wordt gereduceerd tot handelbare deeltjes zodat U steeds enkele uren met Uw Spectrum kunt doormaken en achteraf enkele concrete zaken zult hebben gerealiseerd. Handig daarbij is een verscheidenheid aan programma's die rijkelijk gebruik maken van het uitzonderlijke scala van Spectrumfaciliteiten.
Veel computerliteratuur lijkt het vernuft van haar auteur ten toon te spreiden. Wij doen dit niet: in dit boek is alles ongekunsteld en fundamenteel, wij menen dat een *introductie* tot de computerwereld op deze manier moet worden voorgesteld. De moeilijkste fase is immers het van start gaan.

1. FUNDAMENTEEL BASIC

Als we teruggaan tot in de beginperiode van de computers zien we dat men slechts met de machine kon communiceren d.m.v. lange opsommingen als:
01000111010011010111111110001011000 1 dit is echter onoverzichtelijk. Maar computers *denken* nu eenmaal "in hun binnenste" op deze manier ("0" betekent dan "geen elektrische stroom" en "1" betekent "een bepaalde elektrische stroom"): deze rudimentaire taal is de *machinecode* van een bepaald type van computer. Als door toverkunst stelt deze taal U in staat alles te doen wat U wilt; en meestal heeft ze als voordeel zeer *snel* te lopen ... maar ze is alles behalve praktisch om programma's mee te schrijven
Gelukkig hoeft dit tegenwoordig niet meer omdat men een computer kan *leren* commando's te accepteren in een andere dan zijn "moedertaal". Nú werkt een ijverig programmeur een soort "woordenboek" uit om er een nieuwe taal mee te vertalen in de machinecode, deze nieuwe taal wordt in de computer ingebracht (ofwel via één of andere uitwendige bron, ofwel ingebouwd in de hardware). Het woordenboek noemt men de *compiler*, of de *interpreter* (afhankelijk van de manier waarop het werkt).
De Spectrum beschikt over een ingebouwde interpreter voor een taal die we kennen onder de naam BASIC. dit is het acroniem voor "Beginners' All-purpose Symbolic Instruction Code". Deze taal werd in het midden van de zestiger jaren in de VS ontworpen en wordt zeer veel gebruikt. Ze heeft het voordeel vrij ongecompliceerd te zijn lets als een kruising tussen gewoon Engels en de algebra op school. Een beginnend programmeur moet er maar van uitgaan dat de Spectrum slechts BASIC spreekt (Maar d.m.v de LSR-toets kan men toegang tot de Z 80 machinetaal verkrijgen).

EEN BASICPROGRAMMA

Alvorens we de BASIC-grammatica aanpakken, kijken we even naar een eenvoudig BASIC-programma en gaan we na wat er gebeurt en hoe het werkt. Op deze manier zult U onmiddellijk inzien hoe eenvoudig het allemaal is. Grammatische finesses volgen later.

```
10  PRINT "Verdubbelen"
20  INPUT x
30  LET y = x + x
40  PRINT x, y
50  STOP
```

Door dit programma te bekijken kunt U waarschijnlijk al raden wat het doet, maar eerst moeten er een paar opmerkingen worden gemaakt met betrekking tot de vorm van dit programma.

(a) Het bestaat uit een reeks *regels*.
(b) Elke regel is een "geldige" (d.i. een logische acceptabele) BASIC-*instructie* of *-commando* of *-statement*. (Praktisch gezien betekenen deze drie woorden hetzelfde)
(c) Elke regel begint met een nummer. dat we het *regelnummer* noemen.

(Computers maken dikwijls gebruik van Ø voor nul om verwarring met de letter O te voorkomen.)

Bij deze vanzelfsprekende opmerkingen moeten we enkele verduidelijkingen voegen. Sommige ervan hebben betrekking op *alle* BASIC-interpreters die ook op andere machines worden gebruikt, andere gelden slechts voor de Spectrum.
Over opmerking (a) valt niet veel meer te zeggen behalve dan dat op de Spectrum een programmaregel langer kan zijn dan één regel wanneer hij op een TV-scherm wordt geschreven – de machine kan dat eenvoudig aan.
Details voor (b) zijn afhankelijk van de BASIC-"grammatica" waarop we nog terugkomen.
De reden waarom men regelnummers (c) toekent is de volgende. In het beginstadium van uw programmeurscarrière gaat U de machine mededelen specifieke regels in een programma uit te voeren. De instructie "GO TO 73Ø" zal de machine inlichten over wat ze moet doen, n.l. dat wat het programma op regel 73Ø zegt ... maar de machine moet weten over welke regel het gaat. Zoals we zullen zien heeft het voordelen *niet* zo maar "1, 2, 3, ..." te tellen. Het regelnummer moet evenwel een geheel getal zijn tussen 1 en 9999, dit laatste inbegrepen.
Wanneer de machine een programma doorwerkt, doet ze dit van een gegeven regel naar de daaropvolgende (behalve wanneer het programma inhoudt dat naar een andere plaats moet worden gegaan). Daarom heeft de machine deze regelnummers nodig, zelfs wanneer de programmeur er zelf niet naar verwijst. De Spectrumcomputer zal geen programmaregels aanvaarden die niet voorzien zijn van een nummer. *Vergeet dus geen regelnummers!*

WAT GEBEURT ER ?

Typ bovenstaand programma in. Zo dadelijk zullen we U de ingewikkelde toestanden op het toetsenbord verklaren want U zult ondertussen wel gemerkt hebben dat op elke toets een boel gegevens staan en – sterker – bepaalde toetsen acht verschillende effecten hebben, afhankelijk van de "mode" en de "shift" waarin ze zich bevinden! Maar het is niet nodig dat U alle bijzonderheden in één keer te verwerken krijgt.
We nemen aan dat U de machine net hebt ingeschakeld en de "© 1982 Sinclair Research Ltd." – mededeling heeft gekregen. (Indien dit niet het geval is typt U NEW in, of trekt U de stekker even uit het stopkontakt.)
We nemen bijvoorbeeld regel 1Ø. Toets 1 in (bovenste rij links) en dan Ø (bovenste rij rechts). U zult merken dat deze cijfers onderaan het scherm verschijnen, samen met een flitsende "K", die we de *loper* of *cursor* noemen. Nu drukt U toets P in: het volledige woord "PRINT" verschijnt. Dit is een vernuftige eigenschap van de Spectrum die heel wat tijd uitspaart: complete BASIC-woorden kunnen ingetypt worden door één enkele toets te gebruiken. (U hebt gemerkt dat het woord "PRINT" ook op toets P geschreven staat.)
Nu volgen de aanhalingstekens ". Die vindt U ook op toets P, maar in het *rood*. Dit betekent dat U de SYMBOL SHIFT - toets tegelijk met toets P moet indrukken om de aanhalingstekens te krijgen. Laat nu de SYMBOL SHIFT - toets los. Om de hoofdletter V te zien verschijnen drukt U de CAPS SHIFT - toets én toets V in. Nu laat U de CAPS SHIFT - toets los en drukt U achtereenvolgens de toetsen E, R, D, U, B, B, E, L, E, N in, hierdoor wordt in kleine letters "erdubbelen" uitgeprint. Nu de aanhalingstekens sluiten d.m.v. SYMBOL SHIFT en P zoals hierboven.
Regel 1Ø staat nu onderaan het scherm. (De cursor is nu ook een flitsende "L" geworden.) Om de regel in het programmageheugen te laden, drukt U op de toets ENTER, en regel 1Ø schuift naar boven!

Voor regel 20 gaan we op dezelfde manier te werk: druk 2, 0, I (waardoor "INPUT" wordt geschreven) en X in; daarna ENTER.
Regels 30 - 50 zijn analoog: merk op dat U het "=" - teken verkrijgt door gelijktijdig de toetsen SYMBOL SHIFT en L in te drukken, het "+" teken door gebruik te maken van SYMBOL SHIFT en K, de komma d m.v. SYMBOL SHIFT en N, en STOP met SYMBOL SHIFT en A. Vergeet niet in te voeren d.m.v. ENTER zodra een regel correct is ingetypt.
Indien U deze instructies hebt opgevolgd, zal het hele programma bovenaan het scherm gelist staan, en daar blijft het ook staan zolang U niets doet om verandering in aan te brengen.
Druk dus nu toets R in (dit betekent RUN) en daarna ENTER zodat de computer weet dat U klaar bent met het commando.
Eerst zag U op het scherm iets als RUN [L] met een flitsende L. Dit verdween nadat U ENTER indrukte.
Op het scherm verschijnt nu de mededeling "Verdubbelen". Dit is de respons van de computer op regel 10: PRINT "Verdubbelen". Dit gebeurt vrij vlug en nu wacht de computer totdat *U Uw* deel van regel 20 heeft uitgevoerd; INPUT x. Om U hieraan te herinneren is er een [L] onderaan links op het scherm. De computer verwacht van U dat U mededeelt wat x is.
Dit doet U door een getal in te typen, gevolgd door ENTER.
Probeer

2 (ENTER)

In een flits typt de computer

2 4

uit (en in de benedenhoek een mededeling "9 STOP Statement, 50:1" waarover we ons verder geen zorgen maken – dit betekent slechts dat de opdracht correct werd uitgevoerd.)
Probeer dit nogmaals. Hiervoor typt U RUN, gevolgd door ENTER en daar gaan we weer. Wanneer de machine een waarde voor x vraagt, probeert U eens iets anders. Indien U 756,2912 (de decimale komma is voor de computer net als in de engelse taal een punt) invoert en daarna ENTER indrukt, krijgt U

756.2912 1512.5824

Run dit programma nog enkele keren en probeer telkens andere x-waarden.
Dit moet U ervan overtuigen dat wat U ook voor x invoert, de machine steeds twee getallen uitprint: eerst x zelf en daarna het dubbele van x.
Experimenteer en U zult gelijksoortige resultaten verkrijgen.

HOE WERKT HET ?

We zullen dit niet in detail bespreken omdat dit ons zou leiden tot een uiteenzetting over hardware, machinetaal en dergelijke complexe zaken. Maar hoe zou U, de programmeur, denken voor de machine wanneer ze uw programma runt? Wat gaat er allemaal om in dat nietige silicium-chip brein? Als we ons een klein antropomorfisme mogen permitteren, zou het ongeveer als volgt zijn:

10	PRINT "Verdubbelen"	Wel, wel hier komen enkele
20	INPUT x	instructies die ik moet onthou-
30	LET y = x + x	den.
40	PRINT x, y	Ik vraag mij af of
50	STOP	hij reeds klaar is?

RUN ENTER	In orde. Daar gaan we dan. Wat is de eerste regel? Haal hem uit het geheugen: 10 PRINT "Verdubbelen". Ik moet iets uitprinten. Er is een aanhalingsteken "; dan copieer ik wat volgt ...
Verdubbelen	
	totdat ik nog eens een aanhalingsteken " tegenkom. Dit zegt me met printen te stoppen. Niets meer. Op naar de volgende regel: 20 INPUT x. Hij gaat me een getal mededelen dat ik x moet noemen. Ik geef hem een ...
L	
	cursor om hem eraan te herinneren. OK. Ik wacht. Wat zijn ze toch traag die mensen.
2 ENTER	Ha. x is 2. Wat nu? 30 LET y = x + x. Dit betekent dat ik x bij x moet optellen. dat wil zeggen 2 + 2 berekenen. En noem het resultaat y. Dus y is 2 + 2 = 4 Volgende instructie? 40 PRINT x, y. Ik moet x en y uitprinten. dus
2 4	Volgende regel? 50 STOP. Dat is dus het hele zootje. Beëindig met een mededeling ...
9 STOP Statement, 50:1	

Onthoud dat de machine zich door een opgesomde reeks bevelen werkt en ze uitvoert zoals ze zich aandienen. De machine heeft geen "benul" van wat het programma inhoudt. maar U, de programmeur. U weet dat het getallen verdubbelt.
De opdracht voor de programmeur is nu duidelijk. Indien U een taak hebt die U door de machine wilt laten uitvoeren dan moet U een reeks instructies samenstellen die, wanneer ze worden doorgewerkt, het gewenste effect sorteren.
Om dit te kunnen doen moet U BASIC in al haar details beheersen. Uw Spectrum zal een aantal knappe dingen doen indien U weet hoe U ermee moet converseren. Natuurlijk bestaan er tal van spitsvondigheden – een *goed* programma moet niet alleen dé taak uitvoeren, maar moet bovendien snel, duidelijk en efficiënt zijn. Maar dergelijke finesses zijn voor later, nu komt het er in de eerste plaats op aan programma's te schrijven die *functioneren*

2. REKENKUNDE IN BASIC

Zoals in de gewone algebra gebruikt men in de BASIC-algebra letters die "algemene getallen" voorstellen. In de vaktaal noemt men ze *variabelen* – meer bepaald *numerieke* variabelen – maar eigenlijk betekent dit niet meer dan dat ze geschreven worden als x, y, z, a, b, c enzovoort zodat ze kunnen worden gebruikt om algebraïsche uitdrukkingen als x + y – z mee samen te stellen, die de computer kan uitwerken indien U via het programma aan de computer meedeelt welke waarden x, y en z aannemen. (Bijvoorbeeld indien x = 14, y = 3 en z = 9 dan is x + y – z = 14 + 3 – 9 = 8.)
U kunt op de Spectrum meer ingewikkelde symbolen voor variabelen gebruiken dan alleen maar enkelvoudige letters, hedenk echter dat lange namen meer geheugenruimte in beslag nemen.
Er zijn enkele kleine maar belangrijke verschilpunten tussen BASIC-algebra en gewone algebra. De tekens + (plus), – (min) en / (gedeeld door) blijven hetzelfde. (U kunt het :-teken niet gebruiken voor een deling). Maar de vermenigvuldiging wordt aangegeven d m.v. een sterretje *, zodat 5*7 in gewone algebra wordt geschreven als 5 × 7 of 5.7 en als resultaat 35 heeft. De opstaande pijl ↑ heeft betrekking op de machtsverheffing. Zo schrijft men 2↑3 in de gewone algebra als 2^3, wat zoveel betekent als 2 tot de derde macht en de waarde 2 × 2 × 2 = 8 heeft.
U mag een combinatie van getallen, variabelen en rekenkundige tekens gebruiken en zodoende ingewikkelder uitdrukkingen maken. Bijvoorbeeld

a * x ↑ 2 + b * x + c

schrijft U in de "alledaagse" algebra als $ax^2 + bx + c$. Stel dat de machine weet dat in bovenstaande uitdrukking a = 4, x = 5, b = 3 en c = 7. U zou kunnen denken dat de uitdrukking als volgt wordt uitgewerkt :

a * x = 4 * 5 = 20,

a * x ↑ 2 = 20 ↑ 2 = 400,

a * x ↑ 2 + b = 400 + 3 = 403,

a * x ↑ 2 + b * x = 403 * 5 = 2015,

a * x ↑ 2 + b * x + c = 2015 + 7 = 2022.

Dit resultaat verkrijgt U indien U de uitdrukking van links naar rechts uitwerkt, maar dit komt helemaal niet overeen met de algebraïsche uitwerking van $ax^2 + bx + c$, want dit levert ons

$$4 \times 5^2 + 3 \times 5 + 7 = 100 + 15 + 7 = 122$$

Wat is er dan fout gegaan?
De oorzaak moet worden gezocht in het feit dat in gewone algebra een aantal regels bestaan die bepalen welke operatie vóór een andere operatie moet worden uitgevoerd. In BASIC bestaan gelijksoortige reglementen. In BASIC worden alle ↑'s *eerst* uitgewerkt, daarna alle /'s en *'s en tenslotte alle +'s en –'s, indien er twijfel mogelijk is gebeurt de uitwerking van links naar rechts. Bijgevolg wordt a * x ↑ 2 + b * x + c in werkelijkheid als volgt uitgewerkt :

Eerst de ↑'s : x ↑ 2 = 5 ↑ 2 = 25

Dan de *'s : a * x ↑ 2 = 4 * 25 = 100
b * x = 3 * 5 = 15

Nu de +'s : a * x ↑ 2 + b * x = 100 + 15 = 115
a * x ↑ 2 + b * x + c = 115 + 7 = 122

U ziet dat dit sterk lijkt op gewone algebra.
Zoals in de algebra gebeurt het soms dat U een uitdrukking moet uitwerken die deze regels *niet* volgt. De oplossing in BASIC is dezelfde: U gebruikt *haakjes* ( ). Een uitdrukking tussen haakjes wordt eerst in haar geheel uitgewerkt. Dus

(a * x) ↑ 2

wordt uitgewerkt als (4 * 5) ↑ 2 = 20 ↑ 2 = 400
Er bestaat in BASIC slechts één soort haakjes (niet zoals in de algebra waar naast ( ) ook [ ] en { } bestaan). Deze andere haakjes komen ook voor op het toetsenbord maar kunnen niet worden gebruikt in uitdrukkingen. Dus moet U bijzonder attent zijn indien U haakjes plaatst binnen andere haakjes, bv. wanneer U zaken neerschrijft als (3 + 5 * (a − b)) ↑ 4 waar men in de algebra $[3 + 5(a - b)]^4$ mag schrijven.
Het is vooral van belang dat U begrijpt dat de BASIC-algebra precies hetzelfde is als de gewone algebra maar dat men gebruik maakt van symbolen waarmee men minder vertrouwd is.

EEN WAARDE AAN EEN VARIABELE TOEKENNEN

Hiervoor zorgt het LET-statement (toets L). Indien U het op de juiste plaats gebruikt herkent de snuggere Spectrum het automatisch als LET en niet als L (dankzij een proces dat bekend staat onder de naam "automatische syntaxis controle", dit betekent dat wanneer U deze letter intoetst er op gelet wordt welke betekenis zinvol is). Een programmaregel als

40 LET x = 5

vertelt de computer dat *van nu af aan* de variabele x de waarde 5 aanneemt en wel *totdat een ander programmadeel iets anders aangeeft*. Probeer volgend programma :

```
10 LET a = 4
20 LET b = 3
30 LET c = 7
40 LET x = 5
50 PRINT a * x ↑ 2 + b * x + c
```

Indien alles in orde is, zult U het antwoord 122 bovenaan het scherm zien verschijnen.
Men kan dit resultaat op een eenvoudigere manier verkrijgen: probeer

10 PRINT 4 * 5 ↑ 2 + 3 * 5 + 7

Het rechterlid van een LET-commando kan een uitdrukking zijn waarvan de computer reeds weet hoe ze moet worden uitgewerkt. In bovenstaand programma kunt U bijvoorbeeld regel 50 aanpassen zodat een nieuwe variabele y wordt gedefinieerd met de waarde a * x ↑ 2 + b * x + c :

50 LET y = a * x ↑ 2 + b * x + c

Laat dit statement volgen door

```
60  PRINT y
```

en kijk na of dit inderdaad werkt.

Voorbeeld : Vallende lichamen

Volgens Gallileo valt een lichaam vanuit rust, onder invloed van de gravitatie, in een tijd van t seconden over een afstand van (afgerond) $5\ t^2$ meter. Als we willen weten hoe ver het voorwerp valt in 17 seconden, kunnen we het volgend programma gebruiken

```
10  LET t = 17
20  PRINT 5 * t ↑ 2
```

Als resultaat wordt 1445 (meter) afgedrukt.
Indien U het resultaat wenst voor een andere valtijd, kunt U de eerste regel van het programma aanpassen.
Probeer dit eens voor volgende tijden :

(a) 97 seconden
(b) 3 seconden
(c) 24,5779 seconden

EEN VERBETERING: HET INPUT-STATEMENT

Misschien vindt U het wel vervelend om steeds opnieuw regel 10 aan te moeten passen. Een meer geavanceerd programma gebruikt het INPUT - commando dat de machine opdraagt te vragen welke waarde een variabele aanneemt. Typ

```
10  INPUT t
20  PRINT 5 * t ↑ 2
```

en run dit. De [L] cursor verschijnt : deze geeft aan dat de machine wacht tot U meedeelt wat t is. Typ 17 in en ENTER ; U krijgt nu het antwoord dat we eerst hadden. Om nu de antwoorden voor (a) tot en met (c) te krijgen, moet U slechts op de RUN - toets drukken en de nieuwe waarde voor t meedelen.

PROGRAMMAPROJECTEN

(d) Het volume van een kubus met ribbe x wordt gegeven door x^3 (in BASIC is dit x ↑ 3 indien x de ribbe voorstelt). Schrijf een programma dat x invoert en het volume van de overeenkomstige kubus uitprint.

(e) Een emmer wordt vastgemaakt aan een touw dat opgerold is rond een cilindervormige windas met straal r. Voor deze bepaalde emmer wordt de neerwaartse versnelling, volgens de handboeken over mechanica, gegeven door

$$a = \frac{32}{1 + 3\,r^2}$$

Schrijf een programma dat r invoert en versnelling a uitprint.

(f) In een bepaalde supermarkt kost een pot honig f 5, een pakje bouillonblokjes f 1 en een blikje kattevoer f 2.5. Schrijf een programma dat de totale kostprijs uitprint van h potten honig, z pakjes bouillonblokjes en w blikjes kattevoer.

EEN VOORPROEFJE VAN PROGRAMMEERLUSSEN

D.m.v. de commando's FOR en NEXT kunnen we enkele van voorgaande suggesties uitproberen zonder overbodige arbeid. Over FOR/NEXT - lussen valt een heleboel te zeggen, we zullen hierop enkele pagina's verder uitgebreid terugkomen. Run eerst dit programma zodat U inziet wat ermee kan gedaan worden.

```
10  FOR x = 1 TO 20
20  PRINT x, x * x
30  NEXT x
```

U zal een opsomming krijgen van de getallen van 1 tot en met 20 en een overeenkomstige lijst met de kwadraten 1, 4, 9 ... , 361, 400.
DE FOR/NEXT - commando's sturen de machine steeds weer door een reeks commando's die zich ertussen bevinden (hier is dit slechts regel 20) en zetten variabele x achtereenvolgens op 1, 2, 3, ... totdat het getal 20 bereikt is. Regel 10 legt deze grenzen vast en start de lus : regel 30 stuurt de computer opnieuw door de lus.
Men kan allerlei zaken tabuleren door in bovenstaand programmaatje regel 20 aan te passen. Indien U de derde machten wilt, probeert U

```
20  PRINT x, x ↑ 3
```

Vervang regel 20 door elk van onderstaande statements en let op de verschillen. De kleine veranderingen van de éne regel naar de volgende zorgen ervoor dat de machine de algebra uitwerkt in andere volgorde met verschillende resultaten

```
(g) 20  PRINT x, 1 / x + 1
(h) 20  PRINT x, 1 / (x + 1)
(i) 20  PRINT x, 1 / 1 + x
(j) 20  PRINT x, 1 / (1 + x)
(k) 20  PRINT x, 1 + 1 / x
(l) 20  PRINT x, (1 + 1) / x
```

Merk tenslotte op dat de waarden van de variabelen in deze programmaregels niet door het LET - commando worden toegekend. De machine ontvangt de toekenning in deze gevallen van het FOR - commando.

ANTWOORDEN

(a) 47045 (b) 45 (c) 3020.3658

```
(d) 10  INPUT x
    20  PRINT x ↑ 3

(e) 10  INPUT r
    20  PRINT 32 / (1 + 3 * r ↑ 2)

(f) 10  INPUT h
    20  INPUT z
    30  INPUT w
    40  PRINT 5 * h + z + 2.5 * w
```

Uw programma hoeft niet *precies* hetzelfde te zijn om "juist" te zijn. De keuze van de symbolen voor variabelen ligt bij U. U kunt de algebra ook op een andere manier uitwerken ; in (d) is ook 20 PRINT x * x * x correct en in (e) is er niets op tegen 20 PRINT 32 / (1 + 3 * r * r) te schrijven.

(g) $\frac{1}{x} + 1$ (h) $\frac{1}{x+1}$

(i) 1 + x [werkt eerst 1 / 1 uit, dit is 1, en telt dit op bij x]

(j) is hetzelfde als (h)

(k) is hetzelfde als (g)

(l) $\frac{2}{x}$

3. HET TOETSENBORD

Het Spectrumtoetsenbord is vrij gesofistikeerd en het resultaat van een intoetsing is op een nogal ingewikkelde manier afhankelijk van de context. U zult er wel aan wennen, maar toch zal het een tijdje duren om wegwijs te worden op het toetsenbord.
Twee zaken beïnvloeden het resultaat van een toetsdruk ; dit is de *shift* die wordt gebruikt en de *mode* waarin de computer zich bevindt.

SHIFTS

Er zijn twee shifttoetsen, CAPS SHIFT en SYMBOL SHIFT. Wanneer één van deze toetsen ingedrukt wordt gehouden terwijl een andere wordt ingedrukt wordt het effect van deze laatste toets gewijzigd. *Hoe* deze wijziging gebeurt, hangt af van de mode, dus laat ons even naar de mode kijken.

MODES

De mode is de "inwendige toestand" van de computer en wordt door de cursor aangeduid. Er zijn vijf modes, aangeduid door volgende (flitsende) cursoren :

K	Keyword mode	(sleutelwoord mode)
L	Letter mode	(letter mode)
C	Capitals mode	(hoofdletter mode)
E	Extended mode	(uitgebreide mode)
G	Graphics mode	(grafiek mode)

U brengt de computer op de volgende manier in de gewenst mode :

1. Normaal staat demachine in de "L" - mode.
2. Na een regelnummer of bij het begin van een rechtstreeks ingevoerd commando schakelt de machine automatisch in "K" - mode.
3. Om vanuit "L" - mode in "C" - mode te geraken, drukt U de CAPS SHIFT - toets in en houdt hem ingedrukt. Om terug naar "L" - mode te gaan laat U hem los. (U ziet de "C" slechts indien U de toets CAPS LOCK gebruikt, maar vergeet dit voorlopig maar.)
4. Om vanuit "L" - mode naar "E" - mode te gaan drukt U *én* de CAPS SHIFT - *én* de SYMBOL SHIFT - toets in. Om terug te keren naar "L" - mode herhaalt U deze handeling.
5. Om vanuit "L" - mode over te gaan naar "G" - mode drukt U de CAPS SHIFT - toets en toets 9 (GRAPHICS) samen in. Om terug naar "L" - mode over te stappen doet U dit nogmaals.

HOE WORDT WAT UITGEPRINT ?

We nemen een bepaalde toets van de onderste drie rijen, bijvoorbeeld toets "R". Deze ziet er zo uit :

```
INT
R      <
     RUN
VERIFY
```

De "INT" is groen ; < en "VERIFY" zijn rood.

1. Om "r" te typen duwt U de toets in in "L" - mode
2. Om "R" te typen drukt U op CAPS SHIFT en gaat U over in "C" - mode ; houdt deze toets ingedrukt en duw dan op "R".
3. Om "RUN" te typen drukt U de toets in in "K" - mode.
4. Om "<" te typen moet U vertrekkend vanuit "L" - mode de "R" - toets indrukken en tegelijkertijd de toets SYMBOL SHIFT induwen.
5. Om "INT" te typen gaat U over naar de extended mode "E", door de twee shifttoetsen samen in te drukken. *Houdt de "E" - cursor in het oog* ; indien U de toetsen te lang indrukt kan de machine terug in "L" - mode springen. Laat dan de toetsen los en druk toets "R" in.
6. Om "VERIFY" te typen gaat U, zoals in (5), over naar "E" - mode, *houdt dan de SYMBOL SHIFT ingedrukt* terwijl U op toets "R" duwt.
7. Om het grafiekkarakter dat door de gebruiker wordt bepaald en dat overeenkomt met "R" te verkrijgen (*zie* het hoofdstuk over de door de gebruiker – gedefinieerde – karakters), gaat U over naar "G" - mode en duwt U toets "R" in.

DE BOVENSTE RIJ TOETSEN

Deze verschillen een beetje van de andere. Ze bevatten geen sleutelwoorden ("K" - mode) maar in de plaats daarvan zijn ze voorzien van een grafisch karakter in de vorm van een rechthoekje. Het hoofdsymbool en de twee rode symbolen werken precies op dezelfde manier als voor de toetsen van rijen 2 tot en met 4. De andere items werken als volgt :

1. Om het witte symbool bovenaan te verkrijgen houdt U de CAPS SHIFT ingedrukt. Dit zijn controlekarakters die in werkelijkheid *niet* worden uitgeprint.
2. Om het grafisch symbool te krijgen, gaat U over naar "G" - mode en drukt U de toets in.
3. Om het grafisch symbool in inverse video te krijgen (onderlinge verwisseling van zwart en wit of algemener van INK en PAPER) gaat U in "G" - mode, houdt U de CAPS SHIFT - toets ingedrukt en duwt U op de toets.
4. De kleuren zijn slechts geheugensteuntjes ; 4 is bijvoorbeeld de code voor GROEN (GREEN).
5. Eigenlijk kunt U nog meer doen dan dit : *zie Handboek*, hoofdstuk 16.

4. HEEEEELLLLPPP !

Wat moet U doen indien iets niet helemaal naar wens gaat ?

In het ergste geval trekt U de stekker een seconde uit het stopcontact. Hierdoor wordt het programma dat U heeft ingeschreven uitgeveegd, dus bestaat uw probleem niet meer. Maar meestal zijn er betere manieren.

(a) Indien U een fout in een programma typt: Gebruik de toetsen 5 en 8 waarop pijltjes staan (merk op dat U de toets CAPS SHIFT nodig heeft), verplaats de cursor [L] tot juist voorbij het euvel en wis het uit door de toets DELETE te gebruiken (toets 0 met CAPS SHIFT).

(b) Indien U een programmaregel wilt veranderen die reeds in het programma zit: om deze regel in zijn geheel te schrappen, schrijft U het regelnummer en drukt U op ENTER. Of wel verplaatst U de > cursor op en neer in de programmaregels d.m.v. de toetsen 6 en 7. Daarna drukt U op EDIT waardoor de regel omlaag wordt gebracht zodat er mee kan worden gewerkt. Verder handelt U zoals in (a).
Truukje: het is vervelend de cursor van regel 10 naar regel 530 te verplaatsen met één regel per keer. Daarom zoekt U een regelnummer juist voor regel 530 die U niet heeft gebruikt, bijvoorbeeld 529. Schrijf 529 ENTER en druk dan op EDIT. U zult nu vaststellen dat regel 530 omlaag komt.

(c) Indien het programma vastzit en er niets schijnt te gebeuren: dit is dikwijls het geval tijdens het debuggen (opsporen en verwijderen van fouten in het programma). U wenst zeker niet alles uit te wissen en opnieuw in te typen.

(c1) Indien het programma nog loopt maar U wilt het stoppen, dan drukt U op CAPS SHIFT en BREAK.

(c2) Indien het programma vastzit op een numerieke INPUT - instructie zal BREAK niets uithalen, maar STOP wel. Indien, door een fout, iets anders op deze plaats misloopt, zal een [?] - cursor worden gegeven ten gevolge van een syntaxisfout. Verwijder de fout met DELETE en STOP dan.

(c3) Indien het programma vastzit op een karakter input – [L] cursor: druk op DELETE en *dan* op STOP.

(d) Indien het programma loopt maar niet doet wat men wenst, lees dan de hoofdstukken over DEBUGGING.

5. INVOER / UITVOER

Er valt veel meer te zeggen over het INPUT - commando dan we tot nu toe hebben gedaan. Om te beginnen kunt U meerdere getallen invoeren in één keer. Het programmaproject (f) uit hoofdstuk 2 kan korter als volgt worden geprogrammeerd:

```
10  INPUT h, z, w
20  PRINT 5 * h + z + 2.5 * w
```

Wanneer U dit laat lopen zult U na *elke* van de drie inputs op de toets ENTER moeten drukken. De getallen worden tijdelijk uitgeprint op de onderste rij(en) totdat ze alle drie werden ingebracht.

De komma's die de getallen van elkaar scheiden, zijn ook een aanduiding van de plaatsen waar de getallen worden uitgeprint ; ze gaan afwisselend in de kolommen Ø en 16. Indien U kommapunten (;) van de komma's maakt, zult U merken dat de ingevoerde getallen, zonder spatiëringen, achter mekaar worden uitgeprint. Om spaties te krijgen moet U ze in een INPUT - commando definiëren :

```
10  INPUT h; "□";z"□", w
```

hierin stelt een vierkantje □ een spatie voor.
U kunt ook *prompts* (mededelingen die U eraan herinneren welke invoer vereist is) in een INPUT - statement uitprinten. Dit doet U door de mededeling, als een deel van het INPUT - statement, tussen aanhalingstekens in te sluiten. Vervang bijvoorbeeld regel 1Ø van daarjuist door :

```
10  INPUT "honig", h
12  INPUT "bouillonblokjes", z
14  INPUT "kattevoer", w
```

U kunt ook ingewikkelder combinaties van dergelijke commando's maken door ze in één enkel INPUT - statement te schrijven.

UITVOER

Tot nu toe hebben we, wanneer we uitgevoerde gegevens uitprintten, de computer laten beslissen *waar* er werd geprint. Dit is niet altijd praktisch en dikwijls resulteert dit niet in een mooie bladspiegel. We wijzigen de printpositie d.m.v. PRINT AT.
Het TV - scherm werd omwille van het uitprinten verdeeld in 22 *rijen*, van boven naar onder genummerd van Ø tot en met 21, en in 32 *kolommen* van links naar rechts genummerd van Ø tot en met 31.
Het diagram van fig 12.1 uit het hoofdstuk GRAFIEK illustreert dit. Hier volgt een programma :

```
10  INPUT "rij", r
20  INPUT "kolom", k
30  PRINT AT r, k; "f"
40  GO TO 10
```

Probeer verschillende waarden uit voor de rij- en kolomvariabelen en ga naar waar het f symbool wordt uitgeprint.
Om een specifieke PRINT - positie te selecteren, specificeert U natuurlijk de rij- en kolomgetallen in het commando. Om een berichtje op de onderste rij met een kantlijn van vijf spaties (dit is kolom 4, omdat de nummering begint bij Ø) te verkrijgen schrijft U

```
10  PRINT AT 21,4 ; "Berichtje"
```

Om tot ongeveer in het midden van het scherm te geraken :

```
10  PRINT AT 10,12 ; "Berichtje"
```

Indien U de AT - instructie niet gebruikt, zal de computer automatisch naar "de volgende" positie gaan. Waar dit is hangt af van wat net ervoor werd uitgeprint. Indien het laatste PRINT - commando niet eindigt op " ; " of " , " dan gaat de machine verder naar de volgende rij. Een " ; " stuurt de machine naar de volgende spatie in *dezelfde* rij. Een " , " stuurt ze naar de volgende beschikbare ruimte in de kolom Ø of 16, afhankelijk van welke kolom eerst vrij is.

In INPUT - commando's hebben de dubbele punt en de komma het zelfde effect.
In samenwerking met deze automatische voorzieningen is het TAB - commando erg nuttig indien U gegevens wilt schrijven in georganiseerde kolommen. Dit programma stelt U in staat een gepersonaliseerde telefoongids op te stellen.

```
10  INPUT "naam"; n$
20  INPUT "netnummer"; z
30  INPUT "abonneenummer"; t
40  PRINT TAB 1; n$, TAB 15; z, TAB 25, t
50  GO TO 10
```

(TAB is toets P in extended mode. Het dollarsymbool duidt een *string* aan, hierover volgt later meer.) Run dit programma en voer zaken in als:

Kees	20	44399
Joop	4494	1248
Mieke	1650	9999

Merk hoe de gegevens worden gerangschikt in drie mooie kolommen. (Om te stoppen drukt U op STOP wanneer een nummer wordt ingevoerd of op DELETE en dan op STOP wanneer een karakterstring wordt ingevoerd.)

PROJECT

Schrijf een programma dat 22 getallen invoert en ze diagonaal van linksboven naar rechtsonder uitprint (gebruik iets in de aard van PRINT AT i, i; n). En nu van rechtsboven naar linksonder (PRINT AT i, 21 – i , n).

6. LUSSEN

U kunt de machine een reeks instructies een gespecificeerd aantal keren laten herhalen door de FOR- en NEXT - instructies te gebruiken.
Dit is één van de meest nuttige instrumenten waarover een programmeur beschikt omdat hij variabelen kan gebruiken om er het resultaat van iedere stap mee te wijzigen.
We weten reeds op welke manier FOR/NEXT - lussen het mogelijk maken tabellen met waarden van een functie als x ↑ 3 uit te printen. Nu volgt een iets moeilijkere toepassing.
De faculteitsfunctie n! wordt, zoals elke middelbare scholier weet, gedefinieerd als:

$$n! = n(n-1)(n-2)(n-3)\ldots 3 \times 2 \times 1$$

Om n! te berekenen kunnen we lussen gebruiken. Het idee is als volgt: we berekenen in trappen: 1, 1×2, $1 \times 2 \times 3$, $1 \times 2 \times 3 \times 4$ ... enzovoort totdat het grootste getal dat in de reeks voorkomt n is. In elk geval neemt U het resultaat van de *voorgaande* berekening en vermenigvuldigt het met het eerstvolgende getal. Dit schema is met een lus zeer goed te realiseren.

```
10  LET i = 1
20  FOR k = 2 TO n
```

```
30  LET i = i * k
40  NEXT k
```

Regel 10 stelt een beginwaarde in voor variabele i. Regel 20 beveelt de machine steeds opnieuw de volgende regels uit te voeren waarbij k achtereenvolgens de waarden 2, 3, 4, 5, ... aanneemt. Regel 40 vertelt de machine wanneer het einde van de instructies bereikt wordt en er moet teruggegaan worden naar het begin van de lus. Bij de eerste lusdoorgang neemt de machine de waarde k = 2 en werkt ze regel 30 als volgt uit

$$i = i * k = 1 \times 2$$

Bij de volgende doorgang is k = 3, geworden, dus wordt regel 30 nu uitgewerkt als $i * k = 1 \times 2 \times 3$ (want $i * k$ was na de eerste doorgang gelijk aan 1×2). Dit gaat zo verder totdat de eindtrap wordt bereikt en k gelijk is aan n en regel 30 wordt uitgewerkt als $i = 1 \times 2 \times 3 \times 4 \times \ldots \times n$. Op dit ogenblik weet de machine, dankzij de grenzen die in regel 20 werden vastgelegd, dat de lus beëindigd is.
Om het programma voor de berekening van n! volledig te maken, voegen we nog instructies toe om naar de waarde van n te vragen en om het resultaat uit te printen :

```
 5  INPUT n
10  LET i = 1
20  FOR k = 2 TO n
30  LET i = i * k
40  NEXT k
50  PRINT n; "□Faculteit"; "□is□"; i
```

U krijgt dan als output bijvoorbeeld :

```
6 Faculteit is 720
```

De vierkantjes in lijn 50 duiden spaties aan.

KONIJNEN KWEKEN

Omstreeks 1220 stelde Leonardo van Pisa, bijgenaamd Fibonacci, een interessant vraagstuk m.b.t. konijnen.
Indien een paar konijnen elke maand een nieuw paar jongen krijgt, en indien het één maand duurt alvorens het nieuwe paar produktief wordt, hoe groeit de konijnenpopulatie dan aan als we één paar konijnen laten telen? (Voor de eenvoud stellen we dat de maandelijkse produktie exact regelmatig is en dat elk konijnenpaar bestaat uit een mannetje en een wijfje)
Een tabel is hier welkom :

Maand	Aantal voortbrengende paren	Aantal nieuwe paren

In maand 0 hebben we één telend paar en 0 nieuwe paren, dus hebben we :

0	1	0

In maand 1 hebben we slechts 1 nieuw paar :

1	1	1

In maand 2 hebben we 1 nieuw paar en wordt het vorige paar produktief :

2	2	1

In maand 3 zijn de 3 paren produktief en hebben we 2 nieuwe paren :

3	3	2

enzovoort

4	5	3
5	8	5
6	13	8
7	21	13

Indien we het totaal aantal konijnenparen in maand m gelijkstellen aan f(m) dan hebben we f(0) = 1, f(1) = 2, f(2) = 3, f(3) = 5, f(4) = 8 enzovoort
Laat ons nu algemener redeneren. Stel dat we in maand m, b voortbrengende paren en n nieuwe paren hebben. Dan worden al deze paren volgende maand produktief en leveren ze b + n *voortbrengende* paren ; en de b voortbrengende paren leveren ons b *nieuwe* paren Dus bevat de tabel steeds twee opeenvolgende regels van de vorm :

Maand	Aantal voortbrengende paren	Aantal nieuwe paren
m	b	n
m + 1	b + n	b

Deze tabel toont ons hoe we f(m) d.m.v. een lus kunnen berekenen. Het komt erop neer dat we om van maand m naar maand m + 1 over te stappen, de oude b moeten veranderen in b + n en de oude n in b. Dit gaat als volgt :

```
10  LET b = 1
20  LET n = 0
30  INPUT m
40  FOR t = 1 TO m
50  LET c = b
60  LET b = b + n
70  LET n = c
80  NEXT t
90  PRINT "f(";m;") □ is □" ; b + n
```

Als U dit programma ontleedt, zult U vaststellen dat het de stappen die werden gebruikt om de tabel samen te stellen precies volgt. Regels 10 en 20 leggen de beginwaarden van b en n vast. Regel 30 informeert naar de waarde van m die we voor f(m) wensen Regels 40 tot en met 80 vormen een lus die opeenvolgende regels van de tabel voortbrengt Merk op dat regel 50 de *oude* b - waarde in herinnering brengt en ze c noemt voor gebruik in regel 70.
De getallen f(m) noemt men de *Fibonacci getallen*. Merk op dat de b en n kolommen in de tabellen dezelfde getallen bevatten maar dat deze getallen een regel opgeschoven zijn (waarom ?). U ziet dan ook dat f(m + 2) = f(m + 1) + f(m) ; met andere woorden : elk Fibonaccigetal is de som van de vorige twee.
Welke waarde heeft f(14) ? f(77) ?

GENESTELDE LUSSEN

U kunt ook lussen leggen in andere lussen, en zelfs lussen in lussen in lussen (tot zo ver het geheugen ze kan bevatten).
Veronderstel bijvoorbeeld dat U een tabel met waarden voor de faculteitsfunctie

wenst uit te printen. U kunt een lus gebruiken zoals voor x ↑ 3 in hoofdstuk 2, maar dan hebt U nog een *andere* lus nodig voor de berekening van n! U krijgt dus iets als

```
 5  FOR n = 1 TO 20
10  LET i = 1
20  FOR k = 2 TO n
30  LET i = i * k      binnenste lus      buitenste lus
40  NEXT k
50  PRINT n, i
60  NEXT n
```

Merk op dat de "FOR n / NEXT n" - lus volledig buiten de "FOR k / NEXT k" - lus werd gelegd. Dit moet zo gebeuren omdat de instructievolgorde anders geen zin heeft – het is net als bij het plaatsen van haakjes.
Een uitdrukking [a + (b – 2c)] + d heeft zin, maar [a + (b – 2c]) + d heeft helemaal geen zin. Verwissel maar eens de regels 40 en 60 onderling van plaats en zie wat ergebeurt. Niet erg bruikbaar, nietwaar ? Het probleem met de Spectrum is dat hij ook programma's accepteert en runt waarin foutief genestelde lussen voorkomen ; natuurlijk stellen de resultaten niet voor wat U er van verwachtte. De Spectrum print *geen* foutmelding uit. Wees dus voorzichtig !

STAPGROOTTE

Indien U zonder meer schrijft, FOR i = 1 TO 20, dan neemt de machine voor de variabele i de waarden 1, 2, 3, 4, 5, ..., 19, 20 aan. Dit wil zeggen dat de machine ervan uitgaat dat U werkt met *stappen* van groote 1.
Dit is echter niet verplicht. Door toets STEP te gebruiken kunt U andere stapgroottes instellen. De instructie

```
10  FOR j = –3 TO 3 STEP .5
```

zorgt ervoor dat j de waarden –3, –2.5, –2, –1.5, –1, –0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3 doorloopt.
Op analoge wijze zorgt

```
10  FOR j = 3 TO 4 STEP .01
```

ervoor dat j de volgende waarden doorloopt : 3, 3.01, 3.02, 3.03, .. en verder, met telkens een increment van één honderdste en eindigend met 3.98, 3.99, 4.
Ook decrementen behoren tot de mogelijkheden :

```
10  FOR j = 10 TO 0 STEP –1
```

Zorgt ervoor dat j de getallenrij 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0 doorloopt.

7. DEBUGGING I

Elke programmeur krijgt al snel te maken met programma's – die als ze voor de eerste maal draaien – nauwelijks of niet werken. De bewerking waarmee men fouten (bugs) uit een programma elimineert, noemt men *debugging*. Het is zeer belangrijk

dat men dit systematisch aanpakt omdat de oorzaak van de fout, zelfs in een tamelijk klein programma, niet altijd even gemakkelijk wordt gevonden. Er zijn weinig zaken die meer frustreren als programma's waarin moeilijk te localiseren fouten zitten.
We bekijken eerst even de soorten fouten die kunnen voorkomen. Globaal kunnen we ze in twee groepen indelen : *syntax errors* en *runtime errors*.

SYNTAX ERRORS

Dit zijn fouten die de machine kan identificeren op het ogenblik dat U ze intypt. Stel bijvoorbeeld dat U

```
FOR p = 1 - 7
```

intypt denkend dat het symbool " - " als synoniem voor " TO " kan worden gebruikt. (Het betreft hier dus een fout tegen de grammatica van de taal, vandaar de benaming " syntax " error.)
In dit geval is de Spectrum zeer hulpvaardig voor de beginner. De machine zal U het " - " symbool wel laten typen, maar ze zal zich realiseren dat zich geen omstandigheden kunnen voordoen waarin een volledige programmaregel kan verschijnen als 50 FOR p = 1 - 7, daarom zal ze een [?] - promptsymbool uitprinten (om U mee te delen dat een syntax error werd begaan), bovendien weigert de machine de regel te accepteren zolang het aanstootgevend " - " symbool niet is vervangen.
Op dit punt zullen sommige lezers zich afvragen "Indien de Spectrum weet dat het enige dat na de " 1 " in regel 60 kan verschijnen het sleutelwoord " TO " is waarom zet de machine dan niet op eigen houtje de " TO " neer?"
We moeten hierop antwoorden dat de Spectrum op dat ogenblik nog onvoldoende weet om dit te doen. Er kan bijvoorbeeld nog een cijfer op de " 1 " volgen, zoals in

```
50   FOR p = 12 TO 40
```

(Er zijn nog andere, ingewikkelder mogelijkheden. Het commando

```
50   FOR p = 1 - 7 TO 5
```

is geldig en betekent hetzelfde als 50 FOR p = - 6 TO 5.)
De [?] - prompt kan verschijnen vóór U het einde van de regel bereikt.
Bijvoorbeeld,

```
20   LET e*s=q
```

De machine verwacht na de letter e een " = " symbool of een andere letter of cijfer tegen te komen. dus kan ze de [?] - prompt vóór het " * " symbool neerzetten; hoewel de machine dit pas doet zodra de toets " ENTER " wordt ingedrukt.
De stelregel bij syntax errors is dus : *kijk uit naar de* [?] - *prompt*. Wanneer hij verschijnt zal de reden wel voor de hand liggen: indien dit niet zo is, verifieer dan het statement dat U aan het schrijven bent met behulp van de betreffende paragrafen in het *Handboek*.

RUNTIME ERRORS

Tijdens een programmarun kunnen zich verschillende types van fouten voordoen. Het is nuttiger om specifieke voorbeelden van mogelijke runtime fouten te geven dan ze in algemene bewoordingen te definiëren. Om te beginnen kijken we naar volgend stukje programma.

```
10   FOR p = - 4 TO 4
```

```
20  LET a = 10 / p
30  PRINT a
40  NEXT p
```

Al deze regels zijn perfect BASIC en er zullen geen syntax errors worden gegeven. In werkelijkheid zal het programma perfect beginnen te werken nadat "RUN" wordt ingevoerd en zal het hetvolgende uitvoeren :

−2.5	[d.i. 10 / (−4)]
−3.3333333	[d.i. 10 / (−3)]
−5	[d.i. 10 / (−2)]
−10	[d.i. 10 / (−1)]

Maar dan zal het programma stilvallen met de foutmelding

6 Number too big, 20 : 1

Hier is het probleem nogal duidelijk, zelfs zonder het *Handboek* te hoeven raadplegen. De melding geeft aan dat de machine vastzit op regel 20.
Daarbij komt dat dit statement al verschillende keren zonder bezwaar werd ingevoerd. Dus moet de fout gezocht worden in één van de grootheden die *veranderen*, d.w.z. in de waarde van a of van p. De waarde van p die het laatst met succes werd behandeld is −1, dus is de lopende waarde van p gelijk aan 0. Met andere woorden, de machine probeert 10 door 0 te delen en het resultaat is oneindig (liever gezegd : niet gedefinieerd).
Waarschijnlijk was U reeds tot deze conclusie gekomen vóór dat U bovenstaande analyse had beëindigd, maar we gebruikten dit voorbeeld om de *soorten* aanwijzingen te illustreren waar U naar moet uitkijken wanneer een fout optreedt.

1. Identificeer de regel die de fout teweegbrengt (het getal achter de komma in de melding).
2. Bepaal of dit statement minstens één maal werd uitgevoerd voordat de foutmelding werd voortgebracht. (Indien dit zo is, heeft de moeilijkheid te maken met een bepaalde waarde van één van de variabelen op het ogenblik dat de fout zich voordeed).
3. Maak gebruik van de melding om meer aanwijzingen te verkrijgen. In dit voorbeeld is dit "Number too big" (getal is te groot). Merk op dat de foutmelding niet *precies* zegt wat er gebeurd is (d.w.z. ze rapporteert niet "Poging om te delen door nul"). Dus is het niet altijd voldoende om eenvoudig naar de foutmelding te kijken in de hoop dat deze precies zal verklaren wat er aan de hand is.

Er moeten nog twee zaken over de vorm van de foutrapporteringen van de Spectrum worden vermeld.
Ten eerste beginnen ze met een getal of een letter (hier is dat 6) dit is een identifier die verwijst naar een ingang (entry) uit Appendix B van het *Handboek*. Deze entry kan U in sommige gevallen helpen bij het bepalen wat er verkeerd is gegaan. In dit geval zegt het *Handboek*: "Berekeningen hebben geleid tot een getal groter dan 10^{38}", wat erg veel lijkt op zeggen dat het getal te groot is ...
Ten tweede is er een getal, dat meestal 1 is, aan het eind van de melding na de dubbele punt. Dit verwijst naar "multi-statementregels" waarbij op één enkele regel meer dan één commando staat. Deze technick hebben we tot nu toe niet gebruikt; voor verdere details, *zie* Hoofdstuk 9 over "Sprongen". Bondig uitgelegd gebeurt het volgende : U kan op één regel verschillende statements plaatsen gescheiden door dubbele punten, en dit eindnummer rapporteert in welk van deze statements de

" bug " zit. Dus betekent 20 : 1 "het eerste statement van regel 20" en zou 20 : 3 betrekking hebben op het derde statement.
Dit lijkt (en is ook) erg praktisch, maar de kans op verwarring is erg groot tenzij U voorzichtig bent. De reden hiervoor is dat de Spectrum elke "natuurlijke onderbreking" in de opeenvolging van statements (zoals bij THEN) voor dit doel beschouwt als het begin van een "nieuw" statement. Dus geeft regel :

```
10  IF p / 0 = 2 THEN LET p = p + 1
```

als foutmelding

```
6 Number too big, 10 : 1
```

Maar indien de regel wordt :

```
10  IF p = 2 THEN LET p = p · 0
```

krijgt U

```
6 Number too big, 10 : 2
```

omdat de fout nu in het tweede deel van het statement zit.
Dit verklaart de vreemde mededelingen die U krijgt wanneer *niets* fout gaat. Typ bijvoorbeeld LIST ; U zult volgende melding krijgen :

```
0  O.K., 0 . 1
```

dit betekent

0	Rapportcode 0 : de machine heeft gedaan wat haar werd opgedragen en is geen moeilijkheden tegengekomen.
O.K.	Beknopte vorm van bovenstaande.
0	De machine heeft net "regel 0" uitgevoerd, wat hetzelfde is als zeggen dat het commando geen regelnummer had.
. 1	Het was het eerste statement van die regel.

Het probleem "delen door nul" duikt dikwijls op. En niet altijd op een zo voor de hand liggende wijze als het werd beschreven. Bijvoorbeeld :

```
30  INPUT p, q. r
40  LET a = (p + q - r * 2) / (5 + (p - r) * (p - r) - 2 * q)
```

zal hetzelfde probleem geven indien voor p, q en r respectievelijk de waarden 7, 15 en 2 worden ingevoerd. (Probeer maar eens!) In het algemeen is het nuttig de noemers te onderzoeken en na te gaan voor welke waarde ze nul worden alvorens de som te maken. We zouden bovenstaand voorbeeld als volgt opnieuw kunnen schrijven :

```
30  INPUT p, q, r
32  LET d = 5 + (p - r) * (p - r) - 2 * q
34  IF d = 0 THEN PRINT "Dit kan niet" : GO TO 30
40  LET a = (p + q - r * 2) / d
```

De betekenis van de GO TO - instructie is duidelijk! Regel 34 is een voorbeeld van een multi-statement regel.

DEBUGGING VRAAGSTUK

Om af te ronden krijgt U nu de kans om Uw kennis van wat tot nu toe werd verteld, te testen. Het volgende programma werd geschreven met de bedoeling een reeks positieve getallen in te voeren en hun gemiddelde uit te printen. Indien we het gemid-

delde van 2.4, 8.1, 7 en 14 willen berekenen, voeren we in :

```
2.4
8.1
7
14
-1
```

De " – 1 " aan het einde wordt uitsluitend gebruikt om aan te duiden dat er geen gegevens meer moeten worden ingevoerd en is *geen* deel van de gegevens. (Dergelijke waarde noemt men dikwijls een *delimiter*. indien U naar regel 40 van onderstaand programma kijkt, ziet U hoe het werkt.)

```
10   LET s=0
20   LET c=0
30   INPUT n
40   IF n < 0 THEN GO TO 100
50   LET c=c+1
60   LET z=s+n
70   GO TO 30
100  PRINT "GEMIDDELDE IS □", s / c
```

In de listing zitten enkele fouten. Tracht ze te corrigeren door het *programma in te brengen zoals het daar staat en het te corrigeren*. Vergelijk Uw oplossing nadat U het programma eens hebt laten draaien met het onze dat in hoofdstuk 11 wordt gegeven.

8. TOEVALSGETALLEN (Random numbers)

De RND - instructie brengt een "toevallig" (random) getal voort tussen 0 en 1 dat gelijk kan zijn aan 0 maar niet aan 1. Dit getal is eigenlijk niet echt toevallig, het is een *pseudo*toevallig getal want de getallen herhalen zich elke 65537 keer maar in de praktijk merkt U hier niets van.
Deze getallen kunt U gebruiken in programma's voor spelletjes. Om bijvoorbeeld een dobbelsteenworp te simuleren stelt U vast dat 6 * RND een toevalsgetal geeft tussen 0 en 6 (waarbij 6 niet is inbegrepen). Dus is INT (6 * RND) ofwel 0, 1, 2, 3, 4 of 5 en is 1 + INT (6 * RND) een van de getallen 1, 2, 3, 4, 5 of 6 en dit op goed geluk. Dit is wat een dobbelsteen doet. Om een toevallige speelkaart uit een spel van 52 te halen gaat U op dezelfde manier te werk en gebruikt U 52 * RND, maar U zult wel wat programmeerwerk moeten verrichten om getallen tussen 0 en 51 (inbegrepen) te vertalen in namen als "KLAVERENBOER" of "SCHOPPENTIEN". Met een beetje nadenken zult U daarin echter wel slagen.
U kunt ook toevalsgetallen gebruiken om statistische simulaties uit te voeren. U hebt hiervan in de "kant-en-klare" programma's aan het einde van dit boek een mooi voorbeeld in het programma "DOBBELSTENEN GOOIEN".

9. SPRONGEN (Branching)

Het volgende programma meldt of een gegeven getal even of oneven is.

```
10  INPUT n
20  IF n=2 * INT (n / 2) THEN PRINT "even"
30  IF n < > 2 * INT (n / 2) THEN PRINT "oneven"
```

Hoe werkt dit programma? Even getallen zijn deelbaar door 2. Dus is n even indien n / 2 een geheel getal is. Dit houdt in dat n / 2 hetzelfde is als INT (n / 2) dus als n = 2 * INT(n / 2).
Regel 20 is zodoende een rekenkundige methode om de Spectrum op te dragen het woord "even" uit te printen als het ingevoerde getal "even is", omdat de machine nu eenmaal geen flauw idee heeft wat "even" betekent tenzij we het hem vertellen in de taal die hij begrijpt.
Om het rekenmechanisme te verduidelijken, werken we enkele getallen uit.

n	22	23	24	25	26
n / 2	11	11.5	12	12.5	13
INT (n / 2)	11	11	12	12	13
2 * INT (n / 2)	22	22	24	24	26

Dit moet volstaan om zelfs de grootste scepticus te overtuigen.
Op dezelfde manier is n deelbaar door k – met k en n zijnde gehele getallen – enkel en alleen als n = k * INT (n / k).
Een laatste opmerking betreft het " < > " symbool in regel 30, dit betekent " is niet gelijk aan ".

VOORWAARDELIJKE SPRONGEN

Deze zijn van de gedaante

```
10  IF dit THEN GO TO n
```

waarin n een regelnummer voorstelt. U kunt dergelijke instructies gebruiken om het verloop van een berekening te wijzigen door over te stappen naar een ander programmadeel.
Wanneer U bijvoorbeeld de vierkantswortel van een getal berekent is het belangrijk eraan te denken dat negatieve getallen geen vierkantswortel hebben. U vermijdt foutmeldingen door te handelen als in .

```
10  INPUT n
20  IF n < 0 THEN GO TO 50
30  PRINT n, SQR n
40  STOP
50  PRINT "vierkantswortel niet mogelijk"
```

Bemerk de STOP in regel 40. Waarom is dit nodig? Gooi deze regel eruit en let op wat er gebeurt!
U kunt op dezelfde manier een bescherming inbouwen tegen niet uit te printen bereiken voor a en b in PRINT AT a, b door op volgende manier te werk te gaan :

```
100  IF a < 0 THEN GO TO 1000
```

```
110   IF a > 21 THEN GO TO 1000
120   IF b < 0 THEN GO TO 1000
130   IF b > 31 THEN GO TO 1000
150   PRINT AT a, b ; " * "
160   STOP
1000  een naar uw mening zinvolle respons ...
enzovoort
```

In dit geval veronderstellen we dat a en b in een voorgaand stukje programma een waarde toegekend kregen.
Voeg deze regels toe aan de kop van voorgaand programma :

```
10   INPUT a
20   INPUT b
```

en vraag de machine te printen op 999, –37 (PRINT AT 999, –37) door a = 999 en b = –37 in te stellen.
Laat regel 1000 als volgt luiden :

```
1000   PRINT "Ik ben zo dom niet meer!"
```

Indien U vindt dat de regels 100, 110, 120 en 130 nogal onhandig lijken (en dat zijn ze inderdaad) raadpleeg dan wat verder de paragraaf LOGICA.

VOORWAARDELIJKE TOEKENNINGEN

Deze hebben de vorm : IF dit THEN LET iets. Een alternatief voor het eerste programma uit dit hoofdstuk is :

```
10   INPUT n
20   LET a$ = "oneven"
30   IF n = 2 * INT (n / 2) THEN LET a$ = "even"
40   PRINT a$
```

Er staat een dollarsymbool in a$ omdat dit geen getal is maar een *string* van karakters (zie hoofdstuk 17).
Wat is dit voor een programma ?

```
10   LET s = INT (2 * RND)
20   IF s = 0 THEN LET a$ = "KOP"
30   IF s = 1 THEN LET a$ = "MUNT"
40   PRINT a$
```

Wanneer U een programma schrijft waarin U iets wilt doen dat afhankelijk is van bepaalde zaken die óf gebeuren óf niet gebeuren, denk dan aan de instructie IF . . . THEN . . .

LOGICA

Logica is een onderwerp dat uitgebreid aan bod komt in geleerde verhandelingen. De Spectrum kan de logica waarschijnlijk beter aan dan U en ik. Hij kan, om precies te zijn, statements combineren die voorkomen op de "dit" - plaats van een "IF dit THEN dat" statement ; hij doet dit door de logische operatoren AND, OR en NOT te gebruiken.

Basisregels : p AND q is slechts waar indien p én q waar zijn.
p OR q is waar indien p óf q óf *beide* waar zijn.
NOT p is slechts waar indien p vals is.

De Spectrum werkt de NOT's vóór de AND's uit en de AND's vóór de OR's. Bovendien werkt hij alle andere dingen uit voor hij één van deze uitvoert. Hierdoor heeft U dikwijls haakjes nodig om de volgorde duidelijk te maken.
Een voorbeeld. We kunnen regels 100 tot en met 130 van het programma uit paragraaf VOORWAARDELIJKE SPRONGEN beter schrijven door ze te vervangen door volgende regel :

```
100  IF a < 0 OR a > 21 OR b < 0 OR b > 31 THEN GO TO 1000
```

MULTI - STATEMENT REGELS

U kunt met de Spectrum meer dan één commando op een regel brengen. U hoeft, om dit te doen, de commando's slechts van elkaar te scheiden d.m.v. dubbele punten. We *kunnen* het programma uit hoofdstuk 1 als volgt schrijven :

```
10  PRINT "Verdubbelen"  INPUT x : LET y = x + x
20  PRINT x, y : STOP
```

En eigenlijk had alles op één regel gekund. U kunt deze eigenschap gebruiken om plaats te sparen (er vallen regelnummers weg) of om een programma gemakkelijker te kunnen begrijpen. De voornaamste moeilijkheid is dat U met een GO TO - instructie slechts het begin van een mult-statement regel kunt bereiken. In het algemeen hebben we multi-statement regels vermeden omdat het meestal eenvoudiger is te zien wat er gebeurt indien men ze niet gebruikt. Maar in één geval kunnen ze erg waardevol zijn. We gebruikten dit truukje reeds in Debugging 1 :

```
34  IF d = 0 THEN PRINT "Dit kan niet" : GO TO 30
```

Indien aan een gegeven voorwaarde voldaan is, voert de machine *twee* handelingen uit en wordt het gebruik van een hoop GO TO's vermeden.
Onthoud hierbij dat alle commando's die op één regel voorkomen na "THEN" afhankelijk zijn van het voorwaardelijk IF - statement dat dan "THEN" voorafgaat. Met andere woorden, een commando

```
IF dit THEN dat : iets anders : nog iets anders
```

zal ertoe leiden dat de *drie* zaken (iets, iets anders en nog iets anders) worden uitgevoerd of *dat er helemaal niets wordt ondernomen* (indien de uitspraak vals is).
Multi-statement regels kunnen praktisch zijn, maar ze bevatten ook dikwijls valstrikken. Men schrijft gemakkelijk :

```
10  IF x = 0 THEN GO TO 20 : IF x = 1 THEN GO TO 1000
20  PRINT "x is nul"
enzovoort
```

in de overtuiging dat de machine zal overspringen naar regel 20 indien x = 0 en naar regel 1000 indien x = 1. Dit is niet zo. Indien x = 1 dan wordt regel 10 geïnterpreteerd als :

```
IF x = 0 THEN . .
        GO TO 20 en IF x = 1 THEN GO TO 1000
```

maar

Indien x < > 0 (wat het geval is indien x = 1) THEN . . .
. . . negeer dan de rest van de regel en ga naar de volgende

In dit geval is de volgende regel, regel 20 en dit is juist wat U met dit statement wilde vermijden!
Indien U echter regel 10 vervangt door :

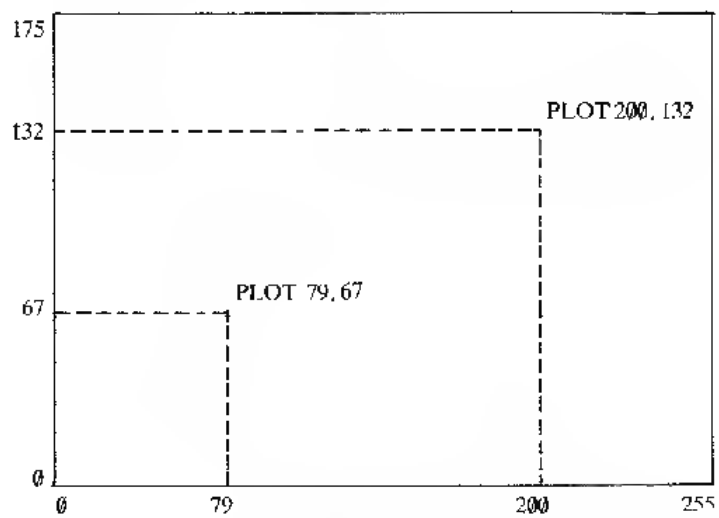
```
10  IF x=0 THEN GO TO 20
15  IF x=1 THEN GO TO 1000
```

dan werkt alles naar wens.

Conclusie : *multi-statement regels zijn erg nuttig in* IF / THEN - *commando's, maar kunnen ook verraderlijk zijn.*

10. UITPLOTTEN

Voor een kleine computer is de Spectrum uitgerust met enkele vrij verfijnde hulpmiddelen voor het maken van tekeningen; hij is in staat grafieken te maken met hoge en lage resolutie (groot en klein oplossend vermogen).
In dit hoofdstuk behandelen we slechts de grafieken met hoge resolutie. Indien U in deze mode werkt, wordt het scherm onderverdeeld in een groot aantal vierkantjes die men "pixels" noemt. Zo zijn er 256 pixels in de horizontale en 176 pixels in de vertikale richting; totaal 256 x 176 = 45056.



Figuur 10.1

Figuur 10.1 toont hoe men naar deze pixels verwijst. De eerste kolom krijgt nummer 0 en de laatste 255, de onderste rij heeft nummer 0 en de bovenste 175. Door regel 50 wordt op het scherm een pixel zwart gemaakt :

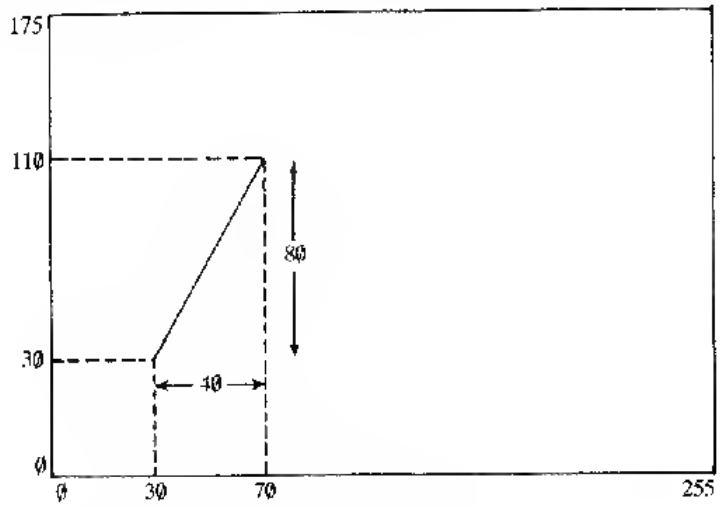
```
50  PLOT 79,67
```

U ziet op het diagram dat de eerste waarde na PLOT het kolomnummer en de tweede het rijnummer geeft.

Het sleutelwoord PLOT deelt het systeem mee dat U wenst te werken met hoge resolutie: U kunt onmogelijk PLOT gebruiken voor lage resolutie-opdrachten. In hoge resolutie zijn er nog twee sleutelwoorden die U kunt gebruiken: DRAW en CIRCLE.
In het eenvoudigste geval wordt DRAW gebruikt om een rechte lijn te tekenen. Het beginpunt van de lijn is in deze instructie stilzwijgend inbegrepen; het is het punt waar zich op dat moment de "tekenstift" bevindt. De twee getallen die op DRAW volgen, geven respectievelijk het aantal kolommen en het aantal rijen die moeten worden doorlopen om aan het eindpunt van de rechte lijn te komen. Dus:

```
10  PLOT 30,30
20  DRAW 40.80
```

brengt de lijn voort die in figuur 10.2 staat afgebeeld.

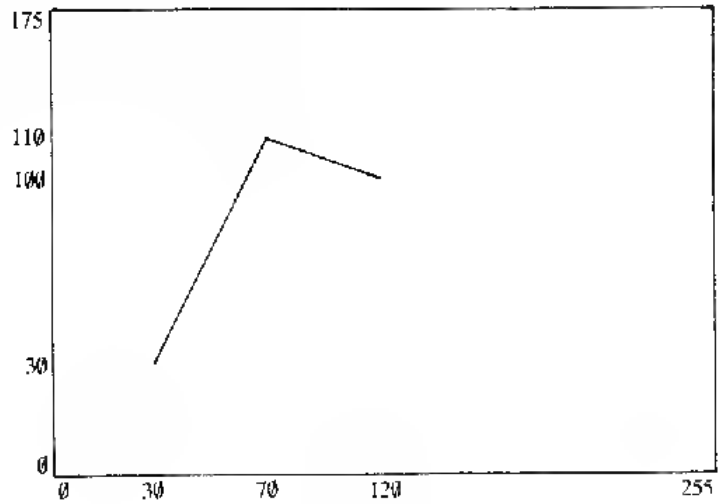


Figuur 10.2

Voegen we

```
30  DRAW 50,-10
```

toe dan ziet U het resultaat in figuur 10.3



Figuur 10.3

Het is dus vrij eenvoudig om bijvoorbeeld rechthoeken te tekenen. Stel dat we eerst de gegevens van de rechthoek invoeren, dit zijn de waarden van de linker benedenhoek, de lengte en de breedte,

```
10  INPUT "Meest linkse kolom" ; lk
20  INPUT "Onderste rij" : br
30  INPUT "Breedte" ; w
40  INPUT "Hoogte" : h
```

Eerst tekenen we de plaats van de linkerbenedenhoek :

```
50  PLOT lk,br
```

Dan tekenen we de basis van de rechthoek met :

```
60  DRAW w,0
```

en de rechterzijde :

```
70  DRAW 0,h
```

de bovenkant :

```
80  DRAW -w,0
```

en ten slotte de linkerzijde :

```
90  DRAW 0,-h
```

Indien we de rechthoek willen opvullen met streepjes (arceren of schaduwen dus) wordt het wat ingewikkelder: we moeten dan eerst alle vertikale lijnen tekenen tussen de linker en de rechter zijde. Stel dat we slechts één van deze streepjes willen tekenen, namelijk het streepje dat zich in de kolom met nummer c bevindt. (Alles wat we op dit moment over c weten is dat c groter is dan lk, de kolom waarin zich de linkerzijde bevindt, en dat c kleiner is dan lk + w, de plaats waar de rechterzijde staat). De code wordt :

```
110  PLOT c, br + 1        [ om de "stift" in de goede positie te brengen ]
120  DRAW 0, h - 1
```

Dit moeten we nu doen voor alle waarden van c van lk + 1 tot lk + w – 1. De +1 en –1 worden aan deze waarden toegevoegd omdat het niet nodig is de opstaande zijden nogmaals te tekenen. Het gebruik van een FOR - lus ligt voor de hand :

```
100  FOR c = lk + 1 TO lk + w - 1
130  NEXT c
```

Om er voor te zorgen dat het hele proces wordt herhaald zodat we streepjes over de volledige rechthoek tekenen, moeten we nog regel 140 toevoegen :

```
140  GO TO 10
```

Natuurlijk zullen hier de streepjes altijd worden getekend omdat regels 100 tot en met 130 steeds worden uitgevoerd. We kunnen het programma aan de gebruiker laten vragen of hij in zijn rechthoek streepjes wil tekenen :

```
48  INPUT "Streepjes tekenen? (ja / nee)" ; b$
```

en als volgt kan de "vulling" worden genegeerd indien het antwoord nee is :

```
95  IF b$ = "nee" THEN GO TO 10
```

Dit alles gebeurt in de veronderstelling dat de gebruiker verstandig te werk gaat en niet probeert lijnen te trekken die buiten de grenzen van het scherm vallen.

We moeten dus enkele tests tussenvoegen die controleren of het te tekenen streepje wel binnen de rechthoek valt, vóór het wordt getekend.
Ten eerste : bevindt de uiterst linkse kolom zich op het scherm?

```
15  IF lk < 0 OR lk > 255 THEN PRINT "dit kan ik niet tekenen" :
    GO TO 10
```

Vervolgens : is de plaatsing van de onderste rij mogelijk?

```
25  IF hr < 0 OR br > 175 THEN PRINT "dit kan ik niet tekenen" :
    GO TO 20
```

Is de breedte niet negatief?

```
33  IF h < 0 THEN PRINT "dit is absurd!" : GO TO 30
```

Past de bovenste lijn wel op het scherm?

```
36  IF br+h > 175 THEN PRINT "bovenste lijn past niet" :
    GO TO 30
```

Is de lengte niet negatief?

```
42  IF w < 0 THEN PRINT "dit is absurd!" : GO TO 40
```

Valt de rechterzijde wel binnen het beeldscherm?

```
44  IF lk+w > 255 THEN PRINT "rechterzijde past niet" :
    GO TO 40
```

Bovendien moeten we ook kijken of als antwoord op regel 48 niets anders dan "ja" of "nee" werd ingevoerd :

```
49  IF b$ < > "ja" AND b$ < > "nee" THEN PRINT
        "Wilt U aub met ja of nee antwoorden" : GO TO 48
```

Hierbij moeten we opmerken dat voor de machine het intikken van "ja" niet hetzelfde is als "JA" intikken. Indien we het programma in de huidige vorm laten staan, zal de machine, indien U als respons op regel 48 "JA" of "NEE" geeft (met de caps lock ingedrukt) in hoofdletters – het volgende blijven herhalen

```
Wilt U aub met ja of nee antwoorden
Streepjes tekenen? (ja / nee)
```

Probeer eens regel 49 zo aan te passen dat de machine de gebruiker toelaat óf kleine óf hoofdletters in te typen (In dit geval is iets als Ja of Nee nog uit den boze!).

CIRKELS BESCHRIJVEN

Cirkels trekken is heel gemakkelijk. Er bestaat een speciaal sleutelwoord, CIRCLE, waarna U de kolom en de rij specificeert van de plaats van het middelpunt en daarna de straal van de cirkel. Dus, bijvoorbeeld :

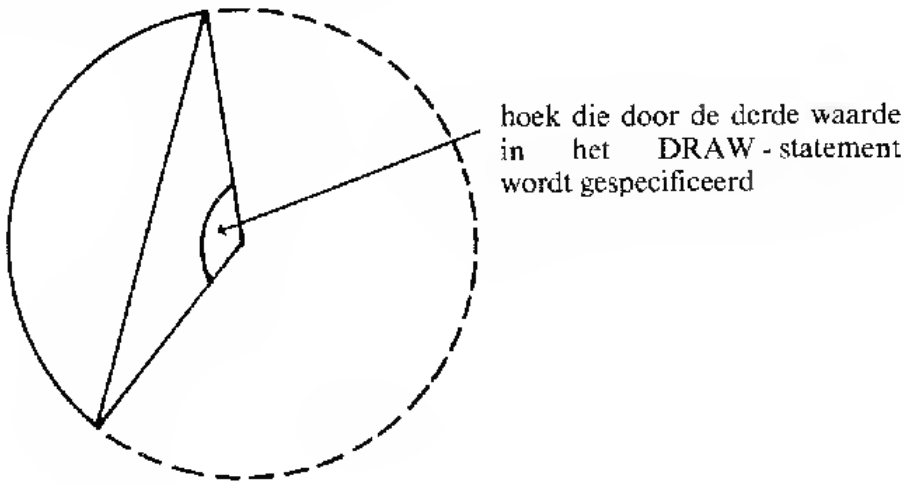
```
20  CIRCLE 50, 70, 30
```

zal een cirkel tekenen waarvan het middelpunt zich bevindt op het snijpunt van kolom 50 met rij 70 en waarvan de straal 30 is.
Maar U kunt ook DRAW gebruiken om cirkels, of beter gezegd stukken van cirkels, te tekenen. Probeer eens :

```
10  PLOT 20, 30
20  DRAW 60, 100
30  DRAW -60, -100, 1
```

U zult zien dat regel 20 een rechte lijn tekent en dat regel 30 terugkeert naar het

oorspronkelijk geplotte punt via een cirkelvormige baan en niet volgens een rechte lijn. De derde variabele na DRAW beveelt de Spectrum een cirkelboog te tekenen; dit gebeurt op de volgende manier.
Stel U de volledige cirkel voor (het stuk dat niet wordt getekend werd in figuur 10.4 met streepjeslijn weergegeven).



Figuur 10.4

Indien U lijnen trekt van het middelpunt naar de uiteinden van de boog dan is de hoek die aldus wordt gevormd de derde waarde van het DRAW - statement. Hierbij wordt deze hoek in het statement niet uitgedrukt in graden maar in radialen.
1 radiaal is iets minder dan 60° (immers de volledige cirkelomtrek die 360° is, heeft 2 π radialen of ongeveer 6,28 radialen: 1 radiaal komt dus overeen met ongeveer

$\frac{360}{6,28} = 57,3°$).

In de praktijk betekent dit dat indien U een kleine hoek kiest (bijvoorbeeld 0.1) U een boog krijgt die bijna met een rechte samenvalt. Bij grotere hoeken komt de cirkelvorm meer tot uiting. Probeer:

```
10  PLOT 60, 20
15  FOR a=0.4 TO PI STEP 0.4
20  DRAW 100, 0
30  DRAW -100, 0, a
40  NEXT a
```

Ziet U hoe de vorm van de boog een halve cirkel benadert wanneer de hoek in de buurt van PI (= 3,14159) komt? (Immers π radialen = 180°!)
Verander regel 15 nu in:

```
15  FOR a=0.4 TO 2 * PI STEP 0.4
```

Langzamerhand krijgt U een volledige cirkel en na een tijdje zult U merken dat deze boog niet meer op het scherm past. Het doet er niet toe hoe kort U de oorspronkelijke rechte maakt. op een bepaald moment komt de boog niet meer op het scherm. De reden is dat voor 2 * PI radialen (360°) dus een volledige cirkel, deze volledige cirkel de oorspronkelijke rechte bevat en dus een oneindige straal heeft

waardoor hij op geen enkel scherm meer past! Zorg er dus voor dat de hoek niet te groot wordt (bij 5 radialen heeft U reeds een grote cirkelboog) en zelfs dan moet U voorzichtig zijn; U raakt makkelijk van het scherm af, en een test om te zien waar U uiteindelijk terecht zult komen is niet zo eenvoudig.

DE GAATJES OPVULLEN

We hebben gezien hoe een rechthoek wordt opgevuld, maar een cirkel of een cirkelsegment opvullen lijkt een moeilijker opdracht te zijn. De reden hiervoor is dat het veel moeilijker is vast te stellen waar de DRAW - instructie moet beginnen en eindigen omdat deze waarden niet vast liggen zoals bij een rechthoek.
We moeten er dus iets op vinden om te weten te komen op welke plaatsen de figuur de rijen snijdt die we willen intekenen. (We zullen, bij wijze van afwisseling, rijen in plaats van kolommen invullen).
Gelukkig biedt de Spectrum ons een mogelijkheid om te weten te komen of een bepaalde pixel werd ingeïnkt, dit is de POINT - functie. Als we schrijven :

```
200  LET g = POINT (20, 30)
```

dan zal g de waarde één krijgen indien de pixel op 20, 30 ingeïnkt werd, en waarde nul in het andere geval.
Het probleem is als volgt samen te vatten :

1. Zorg voor een rechthoekig raam (frame) rond de te schaduwen figuur. In dit raam zoeken we de snijpunten met de figuur.
2. Doe voor elke rij het volgende :
   (a) Zoek van links naar rechts tot de figuur wordt gesneden. Noteer dit punt.
   (b) Zoek van rechts naar links tot de figuur wordt gesneden. Noteer dit punt.
   (c) Trek een lijn tussen beide gevonden punten.

Hier volgt het programma :

```
600  INPUT "frame kolommen" : lk, rk        [ linker en rechter kolom van het frame ]
610  INPUT "frame rijen" ; br, tr           [ onderste en bovenste rij van het frame ]
620  FOR r = br TO tr                       [ van links naar rechts zoeken naar de snijpunten ]
630  FOR c = lk TO rk
640  IF POINT (c,r) = 1 THEN GO TO 660
650  NEXT c
655  GO TO 730                              [ indien het programma hier aankomt, betekent dit: "geen snijpunt op deze rij", ga dus verder naar volgende rij ]
660  LET c1 = c                             [ c1 is de uiterst linker kolom van de figuur ]
670  FOR c = rk TO lk STEP -1               [ van rechts naar links zoeken naar de snijpunten ]
680  IF POINT (c,r) = 1 THEN GO TO 700
690  NEXT c
700  LET c2 = 2                             [ c2 is de uiterst rechter kolom van de figuur ]
```

```
710  PLOT c1,r                    [teken
720  DRAW c2 - c1, 0              in]
730  NEXT r
```

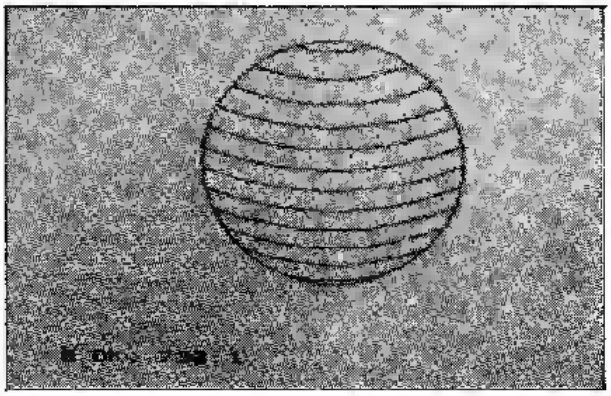
Om iets te zien gebeuren moet U natuurlijk dit stuk programma door een routine laten voorafgaan die een gesloten figuur tekent, bijvoorbeeld de routine bij het begin van vorige paragraaf die een cirkelsegment tekent.
Door enkele eenvoudige wijzigingen aan te brengen wordt dit programma verder geperfectioneerd.
Verander eerst regel 620 in :

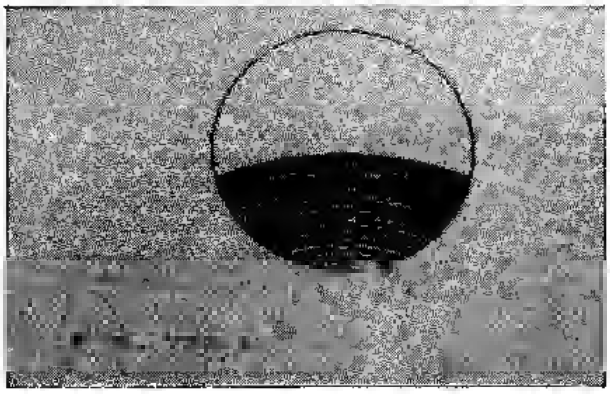
```
620  FOR r = br TO tr STEP s
```

en schrijf dan een regel die de gebruiker een waarde voor s laat invoeren (op een geschikte plaats vóór regel 620). Indien voor s de waarde 1 wordt gekozen is het resultaat hetzelfde als voorheen, maar met s = 2 wordt slechts om de twee rijen een rij ingevuld, in plaats van een schaduweffect krijgen we dan een gearceerde figuur. Met groter wordende waarden voor s liggen de arceringen natuurlijk verder van elkaar.
Schrijf nu regel 720 als :

```
720  DRAW c2 - c1, 0, a
```

en voer a op een geschikte plaats in. Probeer eens een cirkel te arceren met een vrij kleine waarde voor a (bijvoorbeeld 0.5) ; U zult nu een driedimensioneel effect krijgen.
Kies nu de framerechthoek zo dat de bovenste helft van de cirkel er boven uitsteekt. Gebruik voor a dezelfde waarde maar neem voor s opnieuw 1. U krijgt nu een maanoppervlak met schaduwzone.
U krijgt het mooiste effect met een witte figuur (INK 7) op een zwarte achtergrond (PAPER 0).





ELEKTRONISCH VLAKGOMMEN

Het uitgommen van figuren is al even fascinerend als het tekenen ervan. Hiervoor gebruikt U ook de commando's PLOT, DRAW en CIRCLE maar U moet de Spectrum meedelen dat U een vlakgom "in de hand houdt" en geen tekenstift. U deelt dit mee d.m.v. het commando OVER 1.
Probeer dit eens :

```
50  CIRCLE 120, 85. 60
60  OVER 1
70  CIRCLE 120, 85, 60
```

en U zult vaststellen dat een cirkel wordt getekend en daarna weer wordt uitgegomd. Vervang nu regel 70 door GO TO 50 en run nogmaals. De cirkel wordt getekend, dan uitgegomd, daarna opnieuw getekend enzovoort!
Eigenlijk wist de computer in de "OVER 1"- mode iets uit indien er iets is om uit te wissen, maar in het andere geval tekent hij iets. Dus wordt de cirkel getekend als regel 50 voor het eerst werd uitgevoerd, de volgende keer wordt hij uitgeveegd (omdat er iets uit te vegen is), daarna wordt hij getekend (omdat er niets uit te vegen is) enzovoort.
Probeer nu het volgende :

```
10  OVER        0                    [uitschakelen v.d. wisactie]
20  CIRCLE 40. 40, 30
30  CIRCLE 60, 40. 30
40  OVER        1                    [inschakelen v.d. wisactie]
50  CIRCLE 100. 100, 30
60  CIRCLE 120, 100. 30
```

Ziet U het effect? Met de vlakgom uitgeschakeld, worden de eerste twee cirkels getekend die elkaar , zoals verwacht, overlappen. Met de vlakgom ingeschakeld overlapt het andere paar cirkels ook, maar het tweede paar veegt stukjes van het eerste paar uit op de plaatsen waar ze elkaar snijden.
Concluderend werkt CIRCLE (of DRAW of PLOT) in de OVER 1-mode als volgt: "Indien er iets uit te vegen is, doe het dan, teken in het andere geval de gewenste vorm". In bovenstaand programma duurt het effect van de "OVER 0" van regel 10 totdat het tegenbevel wordt gegeven door de "OVER 1" van regel 40. Op de volgende manier kan men een "OVER"-commando gedurende slechts één statement laten werken :

```
70  CIRCLE OVER 0 ; 130, 100, 20
```

De nieuwe cirkel overlapt de vorige zonder de snijpunten uit te vegen maar een daarop volgend statement dat geen OVER-specificering bevat (probeer 90 PLOT 0, 0 : DRAW 150, 148) zal nog steeds worden beïnvloed door de OVER 1 van regel 40 en zal de snijpunten uitvegen. Welk effect krijgt U door in regel 70 "OVER 0 ;" weg te laten?

BEWAREN VAN DE GETEKENDE FIGUREN

Het is praktisch een tekening op tape te bewaren. Zo zouden we een prachtig maanlandschap kunnen ontwerpen en willen bewaren om later in een maanlandings programma te gebruiken.
Met de Spectrum kunnen we de inhoud van het scherm SAVEn en LOADen op precies dezelfde manier als voor gewone programma's. Het enige verschil is dat we nu het woord "SCREEN$" na een normale SAVE of LOAD instructie schrijven om aan te geven dat het de scherminhoud is die we op tape willen laden (of teruglezen) en niet het programma. Voorbeeld :

SAVE "MOON" SCREEN$

zal de inhoud van het scherm als tapefile, genaamd MOON, laden en om het terug op het scherm te brengen :

LOAD "MOON" SCREEN$

11. DEBUGGING II

Aan het eind van hoofdstuk 7 gaven we een programma, dat hieronder wordt herhaald, en stelden we voor het te debuggen.

```
10  LET s=0
20  LET c=0
30  INPUT n
40  IF n < 0  THEN GO TO 100
50  LET c=c+1
60  LET z=s+n
70  GO TO 30
100 PRINT "GEMIDDELDE IS □"; s / c
```

We zullen dit vraagstuk stapsgewijs behandelen. Het is duidelijk dat we eerst proberen of het programma in deze vorm werkt, dus tikken we het op de Spectrum in en runnen we het met enkele eenvoudige gegevens, zoals:

3
7
5
–1 [herinner U dat deze waarde de rol speelt van terminator of delimiter]

het resultaat *zou* 5 moeten zijn. In werkelijkheid krijgt U de foutrapportering "2 Variabele not found, 50 : 1.". Dit betekent dat in regel 50 een variabele werd gebruikt die niet eerder werd gedefinieerd. Aan de hand van de listing zien we dat het hier gaat om i. We geven dus een waarde aan i, bijvoorbeeld:

```
5  LET i=0
```

Op deze manier zal het programma verder werken dan regel 50. Indien we regel 5 toevoegen en opnieuw runnen, stellen we vast dat de mededeling

GEMIDDELDE IS

correct werd uitgeprint. maar dan volgt een foutrapportering:

6 Number too big, 100 : 1

Deze melding hebben we nog gezien: U herinnert zich misschien dat de "6" aangeeft dat de machine heeft gepoogd een stukje rekenkunde uit te voeren dat geleid heeft tot een getal dat groter is dan de machine kan bevatten, en dat de "100" het regelnummer aanduidt waar dit probleem ontstaat.
U mag er dus van uitgaan dat het programma heeft getracht door nul te delen, zoals dit het geval was de vorige keer dat deze foutrapportering opdook. Probeer nog een andere set gegevens.
U krijgt precies dezelfde mededelingen. Welke gegevens U ook invoert, het programma geeft steeds een rekenkundige overflow rapportering.
Waar moeten we dan de fout gaan zoeken? Programma's debuggen is als detectivewerk en een detective zal U vertellen dat als U succes wilt hebben U moet proberen te denken zoals de booswicht denkt, of m.a.w. "de stroper is de beste boswachter". In ons probleem is de Spectrum de tegenpartij, het komt er dus op aan te redeneren

zoals hij. Eerst en vooral moet U Uw gedachtenprocessen vertragen. Dit klinkt misschien eigenaardig want U had de indruk dat computers erg snel waren. Dat is juist, maar de manier waarop ze een probleem "benaderen" is meestal nogal omslachtig. Vervolgens stelt U een model samen van het computergeheugen, of heter van dat deel van het geheugen dat relevant is voor het probleem. In ons voorbeeld werden vijf geheugenelementen betrokken die een naam kregen : s, c, n, i, en z. Een geschikt model zal bestaan uit een tabel waarmee we kunnen tonen hoe de inhouden van deze elementen tijdens de uitvoering van het programma veranderen. Ook kan een aanduiding van het regelnummer waarop een sprong wordt gemaakt erg nuttig zijn. De tabel ziet er dus zo uit :

Regelnummer	s	c	n	i	z	sprong

We hebben een extra kolom "sprong" toegevoegd waarvan we de functie zo dadelijk zullen toelichten. We bouwen de tabel geleidelijk aan op door de uitwerking van elk statement na te gaan. We definiëren een gegevensset, bijvoorbeeld :

2, 1, 4, 6, –1

en stellen systematisch onze tabel samen.

Regelnummer	s	c	n	i	z	sprong
5				Ø		
1Ø *	Ø					
2Ø		Ø				
3Ø			2			
4Ø						x
5Ø		Ø				

* De eerste statements bevinden zich op de regels 5 en 1Ø en doen niets anders dan de geheugenelementen i en s op nul zetten
* Regel 2Ø zet c op Ø
* Regel 3Ø neemt de eerste waarde op uit de gegevensset en plaatst hem in n.
* Regel 4Ø is een "IF"-statement en kijkt na of n negatief is ; dit is niet het geval (n = 2) dus is de test vals, wat we in de "sprong"-kolom aanduiden met een "x", en gaat de sprong naar 1ØØ niet door. Wanneer een test waar is en de sprong vindt plaats dan duiden we dit met een "√" aan in de "sprong-"kolom.
* Regel 5Ø vraagt de machine de inhoud van i bij die van c op te tellen en het resultaat opnieuw in c te plaatsen. Hier moet U reeds achterdocht krijgen, want U weet dat oorspronkelijk i niet werd vastgelegd. Misschien was het wel niet zo'n goed idee om regel 5 toe te voegen.

Natuurlijk heeft het optellen van nul bij nul geen nuttig effect en nu zou U dus echt nattigheid moeten voelen. Indien U zich herinnert dat we typografische fouten zoeken, kunt U vermoeden dat i een typfout is voor 1, iets wat dikwijls gebeurt. Laat ons dus met een nieuwe tabel van start gaan, waarbij we er van uitgaan dat regel 5 overbodig is en dat regel 5Ø moet zijn: 5Ø LET c = c + 1

Regelnummer	s	c	n	z	sprong
10	0				
20		0			
30			2		
40					x
50		1			
60				2	
70					√
30			1		
40					x
50		2			
60				1	
70					√
30			4		
40					x
50		3			
60				4	
70					√
30			6		
40					x
50		4			
60				6	
70					√
30			−1		
40					√
100					

Wanneer het programma regel 100 bereikt zal de mededeling "GEMIDDELDE IS", gevolgd door de waarde van s/c (die nul is!) worden uitgeprint.
Het programma werkt dus nog steeds niet, maar het interessante nu is dat de Spectrum geen foutrapportering geeft; we hebben een nieuwe soort fout veroorzaakt, een fout tegen de logica. De Spectrum kan de gevraagde operaties opperbest uitvoeren, maar juist omdat hij dit doet, levert het ons een foutief antwoord op.
Laat ons even naar de tabel kijken. Het is nu voldoende duidelijk welke functie c heeft. Telkens wanneer een nieuwe waarde in n wordt ingevoerd, wordt de waarde van c met één verhoogd, dus bevat c het totaal aantal ingebrachte waarden, in ons voorbeeld is dat 4. Maar nadat s op nul werd gezet is er met deze variabele niets meer gebeurd en z bevat slechts dezelfde waarden als n maar dan iets later. Misschien moeten s en z in werkelijkheid dezelfde plaats innemen en kunnen we aannemen dat, omdat s het eerst werd vermeld in regel 10, z een typfout is voor s.

Dit betekent dat regel 60 wordt :

60 LET s = s + n

en de tabel wordt de volgende :

Regelnummer	s	c	n	sprong
10	0			
20		0		
30			2	
40				x
50		1		
60	2			
70				√
30			1	
40				x
50		2		
60	3			
70				√
30			4	
40				x
50		3		
60	7			
70				√
30			6	
40				x
50		4		
60	13			
70				√
30			−1	
40				√
100				

Nu krijgen we de uitprinting :

GEMIDDELDE IS 3.25

en dit is correct !

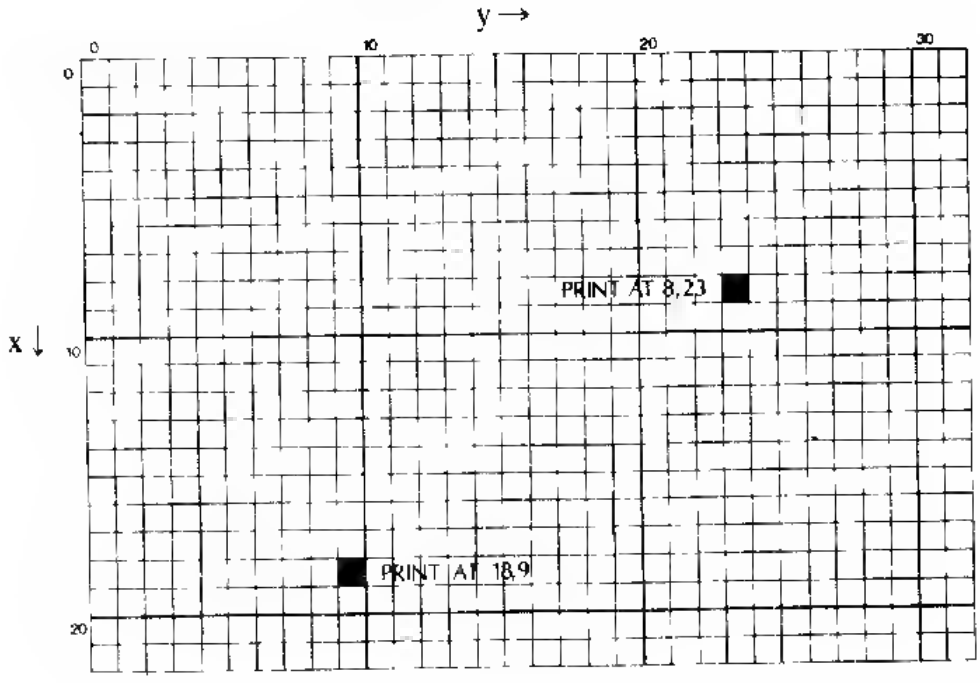
Het proces dat we zopas hebben beschreven noemt men het programma "dry runnen" en op deze manier worden fouten meestal opgespoord. Het is natuurlijk niet altijd nodig alle details in een dry run programmatabel op te nemen (in dit geval zijn de regelnummers bijvoorbeeld overbodig geweest) en zeer dikwijls zal het niet nodig zijn de tabel volledig af te werken alvorens U een licht op gaat.

12. GRAFIEK

Een uiterst aantrekkelijk kenmerk van computers is dat ze tekeningen kunnen maken, en – met voldoende hardware – zelfs mooie en ingewikkelde veelkleurige figuren. Hoewel de Spectrum beperkt is, bezit hij toch voldoende mogelijkheden om U een boeiende introductie in computergrafiek te bezorgen.
Hoofdstuk 10 beschrijft één manier waarop de computer kan worden gebruikt om te tekenen, maar toen ging onze aandacht uit naar academische zaken zoals rechthoeken en cirkels. Door gebruik te maken van de PRINT-faciliteit wordt het mogelijk onderwerpen te tekenen die een grotere visuele aantrekkingskracht hebben.

DE "PRINT"-INSTRUCTIE

De fundamentele instructie is hier PRINT x$, waarin x$ een karakter of een karakterstring voorstelt.
Indien U met de PRINT-instructie experimenteert kunt U vaststellen dat deze instructie U weinig *controle* geeft over de plaats *waar* een bepaald karakter wordt uitgeprint: elke dergelijke instructie wordt door de machine links op het scherm op de volgende beschikbare regel uitgeprint, en dit is niet steeds wat U ervan verwacht.
PRINT AT x, y lost dit probleem op. U moet zich een scherm, ongeveer zoals bij PLOT, voorstellen dat ingedeeld is in vierkantjes die elk twee coördinaten (x en y) toegewezen kregen. Maar er zijn enkele verschillen met PLOT.
Ten eerste zijn de vierkantjes tweemaal zo groot, wat impliceert dat er in elke richting maar half zoveel zijn als er pixels zijn. Vervolgens verschilt het nummeringssysteem van de coördinaten opmerkelijk – het is minder gecompliceerd. Het eerste getal, x, is een *rijnummer* op het scherm; het varieert van Ø bovenaan tot en met 21 onderaan de beschikbare PRINT-ruimte. Het tweede getal, y, is een *kolomnummer* dat de afstand op een rij (d.i. horizontaal) geeft; het varieert van Ø tot en met 31.
Figuur 12.1 toont dit systeem in detail.



Figuur 12.1

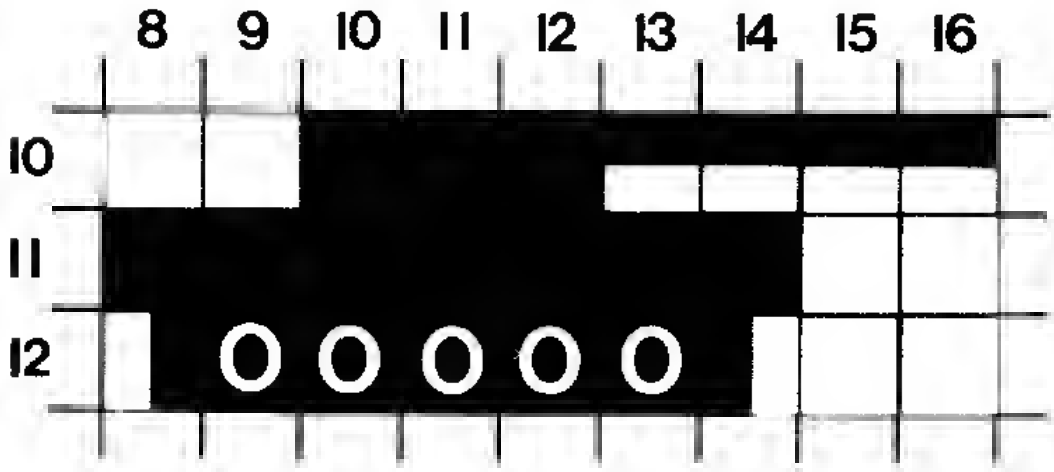
Stel dat u een grafisch karakter ■ in het midden van het scherm wenst uit te printen. U kunt het exacte middelpunt niet bereiken; maar het vierkantje op rij 11, kolom 15 is er niet ver naast. U schrijft dus:

```
10  PRINT AT 11,15; "■"
```

Let op de puntkomma (;) die noodzakelijk is om de computer mee te delen dat hij twee zaken moet doen: locatie 11,15 opzoeken; iets PRINTen.
Door verschillende grafische karakters in elkaar te passen kunt U interessante effecten verkrijgen. De meest directe manier om dit te doen is grote aantallen PRINT AT-instructies gebruiken, maar dan verbruikt U kostbare geheugenruimte. Laat ons voorlopig geen zorgen maken over de efficiëntie en probeer het volgende programma:

```
10  PRINT AT 10, 10; "■■■■■■■"
20  PRINT AT 11,8 ; "■■■■■■■■"
30  PRINT AT 12,8 ; "■■■■■■■■"
```

Wanneer U dergelijke grafische ontwerpen wil uitprinten is het nuttig ze op ruitjespapier te tekenen en de rijen en kolommen te nummeren; op deze manier kunt U de vereiste instructies aflezen. Bovenstaand programma werd op deze manier verkregen uit figuur 12.2.



Figuur 12.2

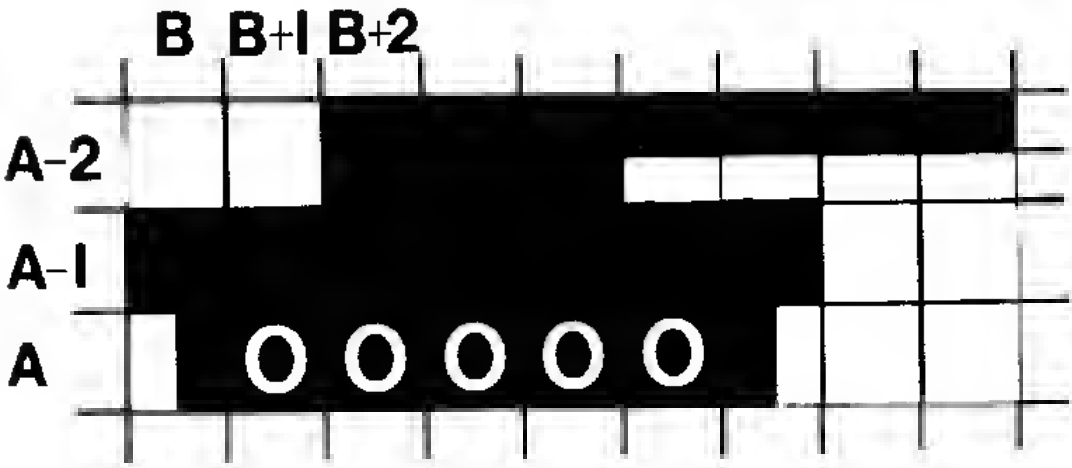
De Spectrum bezit 16 speciale grafische karakters (nummers 128 tot en met 143 van de karakterset-zie *Handboek*), maar U kunt ook andere karakters gebruiken (zoals in bovenstaand voorbeeld). Inverse video (wit op zwart) karakters zijn in dit verband bijzonder nuttig.

DE POSITIE VERANDEREN

Nadat U op een bepaalde plaats van het scherm een tekening hebt uitgewerkt, is het niet moeilijk het programma zo aan te passen dat U dezelfde tekening op een door U gekozen plaats kunt laten uitprinten. Bovenstaande tank (want dat moet het voorstellen) staat op rij 12 met haar linkerkant in kolom 8.
Indien U de tank met de linkerkant in kolom b op rij a wilt plaatsen, moet U slechts de rijen en kolommen van de figuur opnieuw (zie figuur 12.3) nummeren.

Hieruit halen we het nieuwe programma :

```
10  PRINT AT a-2, b+2 ; "■■■■■■■"
20  PRINT AT a-1, b ; "■■■■■■■"
30  PRINT AT a, b ; "■■■■■■■"
```



Figuur 12.3

Om dit programma te doen draaien moet U natuurlijk in een voorgaand stukje programma de waarden van a en b meedelen. Bovendien zal dit programma niet werken, indien het voor a en/of b waarden tegenkomt die de tekening buiten het scherm laten komen. Voor de linkerrand van het scherm betekent dit dat b groter of gelijk moet zijn aan 0 en voor de rechterrand moet b + 9 kleiner of gelijk zijn aan 31, dus moet b kleiner of gelijk zijn aan 22. Analoog moet a groter dan of gelijk zijn aan 2 en kleiner dan of gelijk zijn aan 21.
Het grote voordeel van deze werkwijze is dat we door slechts a en b te specificeren, over het hele scherm tanks kunnen plaatsen. Hier volgen enkele voorbeelden; voeg ze bij bovenstaande regels 10 tot en met 30 (de regelnummers worden door de Spectrum automatisch gerangschikt) en run ze.

```
(a)  1  LET b = 7
     2  FOR a = 2 TO 21
    40  NEXT a
(b)  1  LET a = 15
     2  FOR b = 0 TO 22
    40  NEXT b
(c)  1  LET a = 2 + 15 * RND
     2  LET b = 20 * RND
    40  GO TO 1
```

Zodra U iets over subroutines heeft geleerd, kunt U ook op allerlei manieren gebruik maken van de mogelijkheid op willekeurige posities figuren te tekenen.

BEWEGENDE GRAFIEK

Stel dat U een tank van links naar rechts op het scherm wilt laten bewegen. Bovenstaand programma (b) doet dit, maar laat tijdens deze beweging een spoor na. U kunt dit spoor kwijtraken door blanco spaties over dit spoor te printen maar het eenvoudigst raakt U van dit spoor af door gebruik te maken van een "onzichtbare grens"

Verander regels 10 tot en met 30 op de volgende manier :

```
 1  LET a = 15
 2  FOR b = 0 TO 21
10  PRINT AT a - 2,b + 2 ; "□■■■■■■■■□"
20  PRINT AT a - 1,b ; "□■■■■■■■■□"
30  PRINT AT a,b ; "□■■■■■■■■□"
40  NEXT b
```

Op elke grafiekregel werd een blanco spatie toegevoegd (om te vermijden dat de figuur de rand overschrijdt, zorgen we er in regel 2 voor dat b slechts waarden tot 21 aanneemt).
De begrenzing d.m.v. blanco's is niet op het scherm zichtbaar, maar de manier waarop de computer karakters uitprint brengt mee dat deze blanco's bovenop het vervelende "spoor" worden geprint, waardoor dit spoor wordt uitgeveegd.
Indien U over de mogelijkheid wenst te beschikken de tank achteruit te laten rijden, moet U ook aan de rechterkant zorgen voor een onzichtbare begrenzing. Om ze naar boven te verplaatsen hebben we onderaan een begrenzing nodig en om ze naar beneden te verplaatsen komt de grens bovenaan. Om ze in alle richtingen te kunnen verplaatsen (door overeenkomstig a en b te wijzigen) plaatsen we een begrenzing rond de hele tank, dit betekent dat we twee extra regels blanco's moeten aanbrengen in rijen a - 3 en a + 1

PROJECT

Schrijf een programma dat de tank op het scherm laat bewegen langs de rand van een vierkant met afmetingen 10 bij 10.
Plaats onzichtbare begrenzingen rondom en werk uit hoe a en b moeten veranderen.

BEWEGINGEN STUREN VIA HET TOETSENBORD

Nu we zaken kunnen tekenen die naar hartelust kunnen rondbewegen, is het mogelijk de bewegingen d.m.v. het toetsenbord, dus buiten het programma om, te besturen.
Een onhandige manier om dit te doen is het programma als een lus te schrijven en iets via het toetsenbord in te voeren. Hierdoor wordt alles opgehouden, ook de beweging op het scherm, totdat de toetsen ingedrukt werden.
De INKEY$-instructie is veel beter. Een programmaregel als

```
10  LET c$ = INKEY$
```

laat de machine nakijken welke toets wordt ingedrukt en kent het overeenkomstig karakter toe aan de stringvariabele c$.
Laat ons bijvoorbeeld de tank naar links of naar rechts bewegen door toets 5 voor links en toets 8 voor rechts te gebruiken. (Dit is praktisch voor U omdat op deze toetsen pijltjes staan, maar het is *niet* nodig om de CAPS SHIFT te gebruiken). Hoe gaan we tewerk? We laten de machine de INKEY$ lezen en stellen de printpositie in naargelang deze 5 is of 8.
Dit gebeurt als volgt :

```
 1  LET b = 15
 2  LET a = 12
 3  LET c$ = INKEY$
 6  IF c$ = "5" THEN LET b = b - 1
 7  IF c$ = "8" THEN LET b = b + 1
10  PRINT AT a - 2,b + 2 ; "□■■■■■■■■□"
```

```
20  PRINT AT a - 1, b ; "□■■■■■■■■□"
30  PRINT AT a, b ; "□■■■■■■■■□"
40  GO TO 3
```

Let op de onzichtbare begrenzing aan beide kanten. Het probleem met dit programma is dat als U de toetsen gedurende een te lange tijd indrukt, U van het scherm afraakt. Probeer dit te verhelpen door regels als IF a <0 THEN .. toe te voegen.
Indien U een toets ingedrukt houdt, blijft dit programma lopen en zal de tank blijven bewegen. Soms is dit erg vervelend en dikwijls wilt U dat het programma slechts reageert op veranderingen in de INKEY$ of tenminste op een hernieuwde druk op de toets

```
3  IF INKEY$ < > "" THEN GO TO 3
4  IF INKEY$ = "" THEN GO TO 4
5  LET c$ = INKEY$
...
```

Hierdoor wordt alles op regel 3 vastgehouden wanneer U de toets ingedrukt houdt. Als U de toets loslaat wordt overgegaan naar regel 4 en wordt daar gebleven totdat U op een andere toets duwt of dezelfde toets nogmaals indrukt. Van dan af loopt het programma verder.
WAARSCHUWING: Wees voorzichtig met wat U de machine opdraagt te doen met die INKEY$. Het kan gebeuren dat U toetsen *wenst* in te drukken waarop getallen staan en dat U daarna door middel van de instructie VAL INKEY$ de ingetypte karakterstring wil omzetten in een getal om er rekenkunde mee gaan te bedrijven. Maar om het programma te starten moet U ENTER indrukken en soms leest de Spectrum *dit* als de INKEY$ en probeert deze string in een getal om te zetten. In dit geval zal het programma ontsporen. Of indien geen toetsen werden ingedrukt probeert het de VAL (lege string) te bepalen. U kunt het programma hiertegen beschermen door geschikte IF ... THEN - instructies te gebruiken.

COMBINATIES VAN PRINT EN PLOT

In bepaalde programma's heeft U voor de grafische uitlezing de PRINT en de PLOT-instructie nodig. U moet er wel aan denken dat het resultaat van PRINT AT x, y sterk verschilt van het resultaat van PLOT AT x, y omdat x, y verwijst naar twee verschillende coördinatenstelsels op het scherm. In het Sinclair *handboek* staat een diagram dat beide stelsels voorstelt. Maar meestal kunt U beide soorten commando's op de juiste manier in elkaar laten passen door een ruwe schets op ruitjespapier te maken. Op deze schets duidt U dan het gebied aan waarin U wenst te tekenen en U duidt beide coördinatenstelsels aan. Bij het schrijven van het programma kunt U dan naar deze schets teruggrijpen. Dikwijls is het de moeite waard enige tijd te besteden aan het uitdenken van de programmastructuur *alvorens* U achter het toetsenbord plaatsneemt.

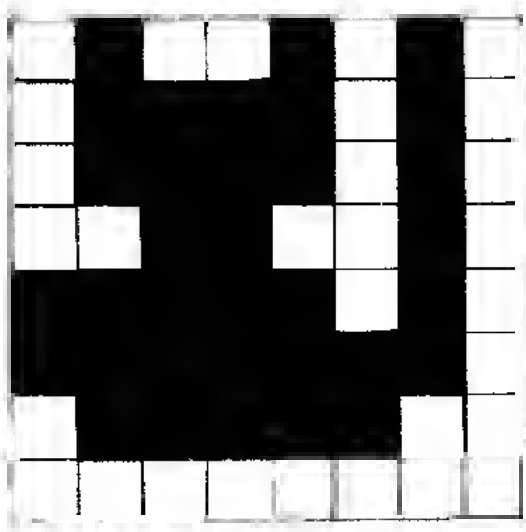
PAUSE

De Spectrum heeft een PAUSE - commando dat de machine een bepaalde tijd laat wachten. Bij bewegende grafiek is dit zeker nuttig om, bijvoorbeeld iets te vertragen dat anders te vlug zou bewegen. Om gedurende n seconden te pauseren typt U

```
PAUSE 50 * n
```

13. DOOR-DE-GEBRUIKER-GEDEFINIEERDE-KARAKTERS

De Spectrum heeft 21 karakters die U naar wens kunt gebruiken. Het zijn de nummers 144 tot en met 164 uit de karaktersverzameling en ze zijn vastgelegd als de letters van A tot en met U. Om deze karakters via het toetsenbord te bereiken moet U overgaan naar "G"-mode en de overeenkomstige letter intoetsen. De techniek om Uw eigen grafische karakters samen te stellen is vrij eenvoudig maar verdient toch verdere verklaring omdat het zo een elegante eigenschap is die van een eerder saai programma toch nog iets speciaals kan maken.
Eerst moet U het gewenste karakters in een rooster van 8 x 8 tekenen, U doet dit door de vierkantjes zwart te maken die bij de uitprinting van het karakter moeten worden be-INKt. Figuur 13.1 toont bijvoorbeeld een "kat" - karakter.



Figuur 13.1

Daarna vervangt U de zwarte vierkantjes door ééntjes en de blanco's door nulletjes. U krijgt dan een lijst als deze :

0	1	0	0	1	0	1	0
0	1	1	1	1	0	1	0
0	1	1	1	1	0	1	0
0	0	1	1	0	0	1	0
1	1	1	1	1	0	1	0
1	1	1	1	1	1	1	0
0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

Vervolgens beslist U welke letter U wilt gebruiken ; de meest voor de hand liggende is "K".
Nu kunt U te werk gaan volgens een omslachtige (maar gemakkelijke) manier door volgende commando's via het toetsenbord in te voeren :

```
POKE USR "K", BIN 01001010
POKE USR "K" + 1, BIN 01111010
```

```
POKE USR "K" + 2, BIN 01111010
POKE USR "K" + 3, BIN 00110010
POKE USR "K" + 4, BIN 11111010
POKE USR "K" + 5, BIN 11111110
POKE USR "K" + 6, BIN 01111100
POKE USR "K" + 7, BIN 00000000
```

POKE plaatst bepaalde informatie in het computergeheugen (zie hoofdstuk 22 over PEEK en POKE), USR deelt de computer mee dat er met door-de-gebruiker-gedefinieerde-karakters wordt gewerkt en BIN staat voor "binair". Belangrijk zijn "K", de letter die U kiest, de getallen 1, 2 ..., 7 die erbij worden opgeteld en de rijen met nulletjes en eentjes die op hun beurt werden gecopieerd van de tabel die U overnam van de kattekening.
Op deze manier kan men elk 8 x 8 patroon van nullen en enen voorzien en de machine zal maximum 21 verschillende door-de-gebruiker-gedefinieerde-karakters opslaan. Ze worden *niet* vernietigd door NEW, ze worden ook niet ge-SAVE-d.
Men kan de karakters ook op andere manieren samenstellen. We schreven voor U een kant-en-klaar programma (KARAKTERBOUWER) dat U toelaat Uw karakter te *ontwerpen* en dat U dan in de computer kunt laden. Eén bepaalde werkwijze zet binaire getallen om in decimale door middel van directe commando's als

```
PRINT BIN 01001010
```

wat 74 als resultaat geeft. De rijen van de kat zijn, in decimale getallen uitgedrukt

```
74, 122, 122, 50, 250, 254, 124. 0
```

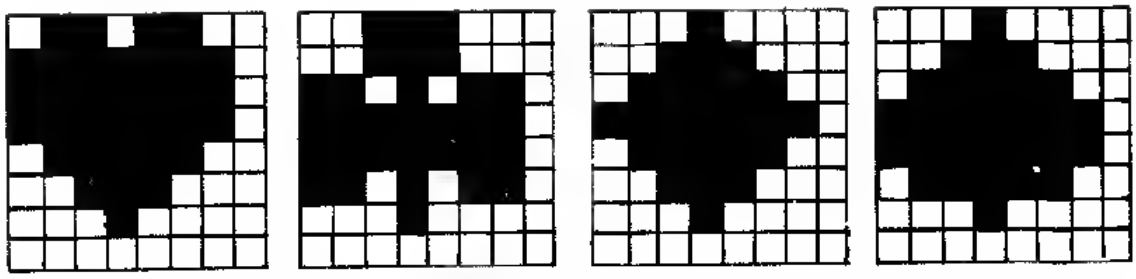
en in plaats van de BINaire getallen te POKEn kunt U deze getallen direct gebruiken :

```
POKE USR "K", 74
POKE USR "K" + 1, 122
.
.
.
POKE USR "K" + 7, 0
```

Nu kunt U de karakters door deze instructies in het programma te schrijven, tijdens een RUN laten verschijnen. Door het programma te "SAVEn", kunt U deze karakters ook bewaren.
Bovendien kunt U de karakters bewaren door die versie van SAVE te gebruiken die een geheugenblok bewaart en door op dezelfde manier te LOADen, maar voor dit boekje zijn dit een beetje te geavanceerde technieken.
U kunt natuurlijk ook de list van getallen, 74, 122 enzovoort, als een array opslaan of het DATA-commando gebruiken (zie hoofdstuk 18 over Gegevens).

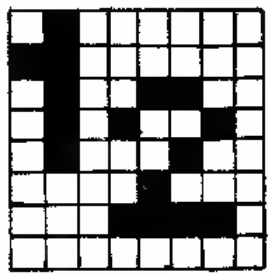
PROJECTEN

1. Ontwerp door-de-gebruiker-gedefinieerde-karakters om er de vier kaartkleuren mee voor te stellen die worden opgeslagen als "H" (harten), "K" (klaveren), "R" (ruiten) en "S" (schoppen). Gebruik hiervoor de roosters van figuur 13.2



Figuur 13.2

2. Maak aan de hand van figuur 13.3 een karakter voor "1/2".



Figuur 13.3

3. Ontwerp een set karakters voor de twaalf dierenriemtekens en plaats ze op posities A tot en met L.

4. Ontwerp twee verschillende karakters die een hond voorstellen maar die slechts verschillen in de stand van de staart. Print ze achtereenvolgens uit op een vaste plaats en laat de staart kwispelen.

5. U kunt een mannetje laten wandelen door verschillende karakters over elkaar te laten vallen.

14. SUBROUTINES

Een subroutine is, in computerjargon, een stuk programma dat op zichzelf kan worden geschreven en herhaaldelijk kan worden gebruikt als deel van een groter programma. Door subroutines te gebruiken schrijft U "meer beschaafde" programma's omdat ze meestal duidelijker laten zien wat er gebeurt. Met subroutines wordt het debuggen van een programma ook eenvoudiger omdat men elke routine afzonderlijk kan debuggen en daarna slechts moet nakijken of ze op de juiste manier met elkaar verbonden zijn.
Het commando GO SUB is hier van toepassing. Het heeft veel weg van GO TO maar het is veelzijdiger en komt meestal op een typische manier voor :

```
100  GO SUB 500
110  verschillende andere zaken
```

```
.
.
.
500  doe iets
.
.
.
570  RETURN
```

hierin stellen de kleine lettertjes, zoals gewoonlijk, andere programmafragmenten voor. De GO SUB - instructie doet hetvolgende :

(a) Als het programma bij regel 100 aankomt springt het naar regel 500, maar het *onthoudt waar het vandaan komt.*

(b) Vervolgens voert het 500 uit en alles wat volgt totdat het regel 570 bereikt waarin wordt meegedeeld terug te keren (RETURN).

(c) Tenslotte keert het terug naar de oorspronkelijke regel (hier is dat regel 100) en voert dan verder de *volgende* instructies uit.

De GO SUB - instructie verschilt van de GO TO - instructie doordat de terugkeer naar het vertrekpunt automatische gebeurt. Toch is het belangrijkste kenmerk van GO SUB de mogelijkheid die U wordt geboden om dezelfde subroutine vanuit *verschillende* regels van eenzelfde programma aan te roepen waarbij het dan bijzonder comfortabel is dat de computer in het oog houdt van welke programmaregel hij komt en dat hij daarna ook naar die regel terugkeert.
Bij wijze van voorbeeld geven we U een gedetailleerd verslag van het schrijven van een programma dat :

1. de toetsen 5, 6, 7 en 8 gebruikt om de cursor [P] over het scherm te verplaatsen.
2. op de plaats van de cursor een willekeurig karakter uitprint dat via het toetsenbord wordt ingevoerd.

Voor de invoer van dit karakter zullen we de INKEYS moeten gebruiken. We lezen door deze instructie de ingedrukte toets in en kennen hem toe aan een stringvariabele a$. We wensen dat het programma slechts reageert op nieuw ingedrukte toetsen zoals we eerder hebben beschreven; dus hebben we een stukje programma nodig zoals :

```
1000  IF INKEY$ < > " " THEN GO TO 1000 ┐ dit wordt
1010  IF INKEYS = " " THEN GO TO 1010   ├ de
1020  LET a$ = INKEY$                   ┘ subroutine
```

We gebruiken dergelijke grote getallen voor de regelnummers omdat dit een subroutine wordt, die we vooral goed uit de weg willen plaatsen (hoewel U eigenlijk dikwijls enkele programmaregels kunt uitsparen door alle subroutines *vooraan* te plaatsen en de run op gang te brengen door GO TO in plaats van RUN te gebruiken). Om uit de routine te raken hebben we volgende extra regel nodig :

```
1030  RETURN
```

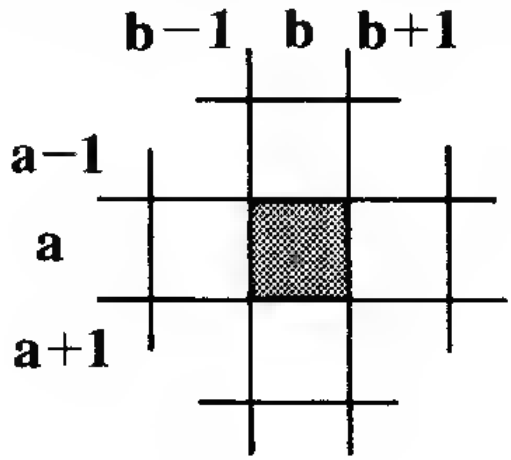
Wat hebben we vervolgens nodig om de cursor [P] her en der te verplaatsen? We zullen beslist moeten weten *waar* hij moet worden uitgeprint, dus hebben we twee variabelen, a en b, nodig die de rij en de kolom aanduiden waar [P] moet worden uitgeprint. Om te vermijden dat alles spaak loopt voor we goed en wel vertrokken zijn, moeten we aan a en b waarden toekennen. Het midden van het scherm lijkt ons een goede startpositie :

```
10  LET a = 10
20  LET b = 15
```

Nu willen we de toetsen 5, 6, 7 en 8 gebruiken om a en b te wijzigen en aldus de cursor te verplaatsen. Bovenstaande subroutine reageert op het toetsenbord, dus ligt het voor de hand wat de volgende stap is :

```
30  GO SUB 1000
```

Nu zal a$ ons vertellen welke van de toetsen 5, 6, 7 of 8 werd ingedrukt. We willen [P] verplaatsen in de richting van de pijltjes op deze vier toetsen. (Daarom gebruiken we ook deze toetsen: de pijltjes zijn slechts mnemonische hulpmiddeltjes!)



Figuur 14.1

Figuur 14.1 toont ons de posities a, b en de vier aangrenzende posities. Uit de figuur halen we de acties van onze vier toetsen :

toets 5:b verandert in b – 1 en laat a ongewijzigd,

toets 6:a verandert in a + 1 en laat b ongewijzigd,

toets 7:a verandert in a – 1 en laat b ongewijzigd,

toets 8:b verandert in b + 1 en laat a ongewijzigd.

Hier volgt een manier om dit te realiseren :

```
40  IF a$ = "5" THEN LET b = b – 1
50  IF a$ = "6" THEN LET a = a + 1
60  IF a$ = "7" THEN LET a = a – 1
70  IF a$ = "8" THEN LET b = b + 1
```

Nadat we de cursor hebben verplaatst, zouden we graag kunnen zien waar hij is en voor de aardigheid laten we hem flitsen :

```
80  PRINT AT a, b; FLASH 1; "P"
```

Wat nu? We willen een karakter invoeren dat moet worden uitgeprint op de plaats van deze [P]. Dus opnieuw de subroutine!

```
90  GO SUB 1000
```

Deze keer leest de machine het toetsenbord in en kent de gevonden waarde toe aan a$ (deze is wat we intoetsten: a, b, c, d, ..., 7, 3, >, ...) en onmiddellijk daarna keert

de machine terug *naar de regel die volgt op regel* 90. Deze regel zal het gevonden karakter uitprinten :

```
100  PRINT AT a,b ; a$
```

Nu zijn we bijna klaar. Tot nu toe werkt het hele programma slechts éénmaal. We moeten teruggaan naar het begin en opnieuw starten maar we wensen de nieuwe waarden van a en b te behouden, dus mogen deze waarden *niet* worden gereset zoals in regels 10 en 15. Er komt dan :

```
110  GO TO 30
```

(regel 30 stuurt de nieuwe waarden onmiddellijk door naar 1000. Waarom zou 110 GO SUB 1000 niet werken?)
Schrijf bovenstaande regels in (we hebben ervoor gezorgd dat de nummers in de goede volgorde staan, dit is niet altijd het geval wanneer U een programma ontwerpt, laat U dus niet misleiden!) en druk op RUN. Er zal niets gebeuren, maar indien U op 5, 6, 7 of 8 drukt zult U de [P] cursor op zijn *nieuwe* positie zien verschijnen. (Indien U een andere toets indrukt, verschijnt hij op de oude a,b - positie – dit is een voordeel van het programma dat we eigenlijk niet hadden gepland) Duw dan op een toets, bijvoorbeeld t De [P] verdwijnt en wordt vervangen door een t. Verplaats de (nu onzichtbaar geworden) cursor d.m.v. 5, 6, 7 of 8 en print uw volgend karakter uit, enzovoort. U kunt op deze manier woorden over het hele scherm schrijven. (Misschien een suggestie voor het opstellen van kruiswoordraadsels met computerassistentie?)
Om te vermijden dat alles aan de randen van het scherm fout loopt, is een stukje bescherming geen slecht idee :

```
75  IF a < 0 OR a > 21 OR b < 0 OR b > 31 THEN GO TO 30
```

en ongetwijfeld zult U nog andere verfijningen kunnen bedenken.
Tot slot nog een voorbeeld: een introductie tot "Computerkunst". Dit programma tekent volledig willekeurig vierkantjes, die zwart of geruit zijn, totdat de geheugenruimte is opgebruikt, of totdat U alles stil legt d.m.v. BREAK.

```
  10  LET a = 10 * RND
  20  LET b = 10 * RND
  30  LET q = 5 * RND
  40  LET r = 5 * RND
  50  LET k = INT (2 * RND)
  60  IF k = 0 THEN LET m$ = "■"
  70  IF k = 1 THEN LET m$ = "▚"
  80  GO SUB 1000
  90  GO TO 10
1000  FOR i = a TO a + q
1010  FOR j = b TO b + r
1020  PRINT AT i, j; m$
1030  NEXT j
1040  NEXT i
1050  RETURN
```

PROJECT : DOBBELSTEEN

Breng door middel van INT (1 + 6 * RND) een randomgetal n voort tussen 1 en 6.

Print dit getal n als de n ogen van een dobbelsteen op het scherm. Herhaal dit als ENTER wordt gedrukt.
Het programma vereist een subroutine voor ieder van de zes mogelijkheden. Indien de subroutine voor n start op regel 500 + 100 * n (dus voor 600, 700, etc.) en eindigt voor de volgende subroutine start, kunt U door middel van GO SUB 500 + 100 * n ingewikkelde conditionele sprongen vermijden. Maar U hebt toch steeds zes aparte RETURN - commando's nodig.

15. KLANK- EN LICHTSPEL

Probeer dit eens :

```
10  FOR i=1 TO 6
20  INK i
30  CIRCLE 100,80,i*10
40  BEEP 0.5,i
50  NEXT i
```

Misschien is dit programma niet zo sensationeel, maar het illustreert toch enkele belangrijke dingen. Ten eerste stellen we vast hoe de kleur van een uitgeplotte (of uitgeprinte) figuur kan worden gewijzigd. Het volstaat door middel van een getal een bepaalde kleur in een "INK" - statement te specificeren.
We hoeven de getallen die bij bepaalde kleuren horen niet te onthouden omdat de kleuren op het toetsenbord boven de bijhorende getallen vermeld staan. Dus, wordt in het programma i = 1 wanneer regel 20 de eerste keer wordt uitgevoerd en wordt het statement "INK 1" door de Spectrum geïnterpreteerd als "gebruik tot nader bericht blauwe inkt voor het uitprinten en uitplotten". Natuurlijk wordt slechts de kleinste cirkel uitgeplot terwijl de "inkt" blauw is. Bij het uitplotten van de volgende cirkel is regel 20 veranderd in "INK 2" en wordt dus rood gebruikt, enzovoort.
Nadat alle cirkels werden getekend laat de computer een triomfantelijk "gebeep" horen, dat het gevolg is van regel 40. De eerste waarde na BEEP geeft de duur van de toon aan in seconden (dus elke beep duurt in dit programma 0.5 seconde) en de tweede bepaalt de muzieknoot die wordt "gespeeld". Indien de waarde nul is dan is de noot een C (do), 1 is C#, 2 is D, 3 is D#, 4 is E, 5 is F (omdat E# niet bestaat) enzovoort. Negatieve waarden geven de noten die op de toonladder lager staan dan C, – 1 is B, – 2 is A# enzovoort.
De eenvoudigste toepassing van BEEP is die waarbij een geluidssignaal de gebruiker opmerkzaam maakt op iets dat is gebeurd (bijvoorbeeld een fout in het programma) of dat de computer op U wacht of dat hij reeds een bepaalde input heeft aanvaard. Men kan er natuurlijk ook muziek mee spelen maar dat reserveren we voor later (zie kant-en-klare-programma's) en in de plaats daarvan zullen we enkele kenmerken van het kleurensysteem bespreken.
Eerst en vooral kan men niet alleen de kleur van de inkt veranderen, ook de achtergrond (die men in BASIC om practische redenen PAPER noemt) kan één van de acht kleuren aannemen. Indien INK én PAPER van dezelfde kleur zijn, ziet U na-

tuurlijk niets gebeuren en dit kan vrij verwarrend voorkomen indien U na een programmarun, die de INK - kleur hebt veranderd in de PAPER - kleur, LIST intypt. (We vermelden dit omdat dit dikwijls gebeurt, waarna men zich dan zit af te vragen waarom het programma werd uitgevoegd. Het werd helemaal niet gewist, men kan het alleen niet zien.)
U zou kunnen verwachten dat U na het schrijven van: "PAPER 2" een rode achtergrond krijgt. Maar indien U dit statement op, bijvoorbeeld regel 5, invoegt zal na RUN niets veranderen. De reden is dat het systeem de kleur van de achtergrond niet kan veranderen indien er al op werd geschreven en de machine kan er onmogelijk zeker van zijn dat nog niets werd geschreven zolang ze geen CLS (clear screen) instructie heeft ontvangen. De machine zal dus pas reageren op de meest recente PAPER - instructie nadat ze een CLS heeft tegengekomen.
Bovendien kan men de kleur van de rand veranderen (door bijvoorbeeld BORDER 4 in te typen wordt het scherm met groen omlijst) en deze instructie heeft haar uitwerking zodra ze wordt tegengekomen. Indien U dus invoert:

```
25  BORDER i
```

dan zal de kleur van de boord altijd die van de uitgeplotte cirkel zijn.
Schrijf regel 25 als:

```
25  BORDER 7 - i
```

Dit is een beetje afstotend (om niet te zeggen smakeloos) nietwaar?
Voeg nu volgende regels toe:

```
60  INK 4
70  PLOT 100,0
80  DRAW 0,175
90  PLOT 0,80
100 DRAW 255,0
```

en RUN het programma.
Zoals U verwacht had, werden twee groene lijnen kruisvormig getekend, maar indien U nauwkeurig kijkt zult U vaststellen dat alle cirkels in de omgeving van de kruisende rechte lijnen groen werden gekleurd. Dit is geen defect in uw televisietoestel en geen storing in de Spectrum, maar een eigenschap van de manier waarop de Spectrum de display realiseert. (In het computerjargon is een gebrek dat niet kan worden verholpen altijd een "eigenschap". Hierdoor klinkt het alsof het zo bedoeld was.) De reden van dit opmerkelijk gedrag is dat de kenmerken van een punt op het scherm (dit zijn de kleur, de helderheid, en het al dan niet flitsen) niet beperkt zijn tot juist één pixel. Ze hebben betrekking op één volledig karakter en dit bevat, zoals we reeds hebben gezien, 64 pixels, gerangschikt in een vierkantje van 8 × 8. Wanneer U dus de kleur van één pixel verandert, veranderen alle andere pixels uit het 8 × 8 gebied die be-INKt zijn mee.
Meestal is dit niet zo erg en indien U de helderheid van een karaktervierkantje verandert, geeft dit meestal ook geen problemen.
De helderheid kan men wijzigen d.m.v. het statement BRIGHT 1 dat de helderheid verhoogt en d.m.v. BRIGHT 0 dat ze opnieuw verlaagt. Wanneer een BRIGHT 1 - commando werd uitgevoerd, wordt alles helder uitgeplot totdat het volgende BRIGHT 0 wordt bereikt. Indien we dus de twee binnenste cirkels helderder wensen uit te plotten dan de andere, kunnen we de volgende statements toevoegen:

```
5   BRIGHT 1
23  IF i > 2 THEN BRIGHT 0
```

Met het flitsen is het heel anders gesteld. U kunt het flitsen in- en uitschakelen d.m.v. FLASH 1 en FLASH 0, dus ongeveer zoals bij BRIGHT. Indien U het FLASH 1-commando toepast in het programma dat cirkels trekt, zullen grote blokken van het scherm op een enerverende manier beginnen aan en uit te flitsen.
Probeer maar eens met :

```
4 FLASH 1
6 CLS
```

en U merkt meteen wat ik bedoel (Net als PAPER vergt FLASH een CLS om te worden geactiveerd). Misschien kunt U deze manier van display ergens gebruiken, maar de enige toepassing die ik kan bedenken is er één voor het opwekken van migraine bij proefratten.
Daarom wordt FLASH meestal gebruikt in PRINT-statements waarbij we hele karaktervierkantjes willen laten aan- en uitflitsen, en niet in PLOT-, DRAW- en CIRCLE-statements omdat de effecten daar niet erg meevallen.
Het is zelfs wenselijk de commando's INK, PAPER, FLASH enzovoort aan een PRINT-statement toe te voegen, waardoor hun effect wordt beperkt tot de door dat bepaalde statement in display gebrachte symbolen en in dit geval moeten ze niet d.m.v. een CLS worden geactiveerd. U kunt bijvoorbeeld het volgende schrijven :

```
110 PRINT AT 10,0 ; INK 5 ; FLASH 1 : PAPER 4 : "x x x"
```

en dan runnen (U moet wel eerst regel 4 schrappen en FLASH 0: CLS als direct commando uitvoeren om van het flitsen af te raken) en U zult merken dat slechts "x x x" aan- en uitflitst. Indien U nu LIST intypt zult U merken dat de inkt nog steeds groen is (dit werd door regel 60 ingesteld), alhoewel regel 110 de kleur op cyaan instelt en de listing niet zal aan- en uitflitsen.
We hebben niet alle mogelijkheden besproken waarop U de display van de Spectrum kunt beïnvloeden. Dit boek is bedoeld als een introductie en het ligt dus niet in onze bedoeling U op te zadelen met alle denkbare details. Wanneer U deze technieken onder de knie hebt, zult U wat graag de rest van het *handboek* doornemen en normaal gesproken zult U hierbij niet op al te veel moeilijkheden mogen stoten.
Tot slot nog drie kleine puntjes. Ten eerste betekent het wissen van het scherm vóór bepaalde eigenschappen effect hebben niet noodzakelijk dat U CLS moet invoeren. Het indrukken van de ENTER-toets waaraan niets voorafgaat, zal hetzelfde effect hebben als CLS : LIST. Dus inplaats van FLASH 0:CLS te typen om het flitsen te doen ophouden kunt U net zo goed FLASH 0 intypen en daarna de ENTER-toets tweemaal indrukken daar waar anders eenmaal voldoende is.
Ten tweede zult U onderhand al wel hebben gemerkt dat de zone van de rand die zich juist onder de achtergrond bevindt, niet altijd verandert zoals zou moeten en de kleur bij het begin van een run behoudt. Misschien hebt U al opgemerkt dat deze strook overeenkomt met de commando- en de mededelingsregel (d.i. de plaats waar het BASIC-systeem met U communiceert). Dit is het probleem : het BASIC-systeem weigert in de val te lopen waarover we het reeds hadden bij het schrijven met groene inkt op een groene achtergrond. Per slot van rekening is het nogal belangrijk dat U weet wanneer de machine U iets probeert duidelijk te maken.
In werkelijkheid varieert de kleur van deze strook tussen zwart en wit, naargelang de kleur van de rand zodat het resultaat op de beste manier uitkomt. Indien U dit programma in een lus legt door

```
120 RUN
```

in te voeren en dan op verschillende plaatsen BREAK (onderbreking) gevolgd door CONT (verdergaan) in te voeren, zal U het effect duidelijk worden.
Ten derde schakelt U de FLASH-eigenschap in wanneer U "FLASH 1" schrijft

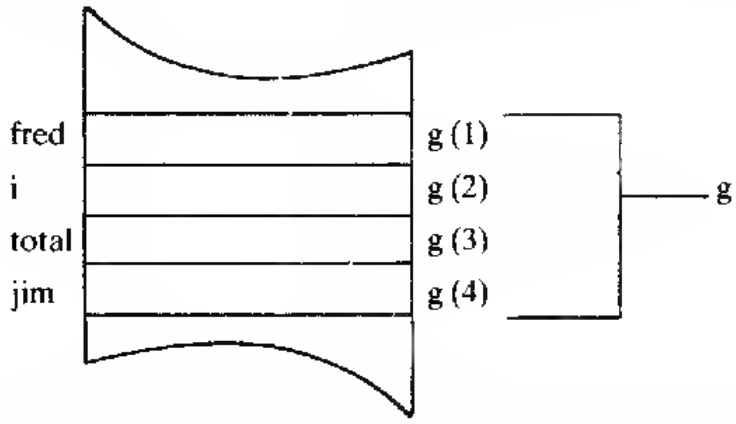
Voor de stijl zou het veel mooier zijn indien U kon schrijven "FLASH aan" en "FLASH uit" of "BRIGHT aan" en "BRIGHT uit", in plaats van deze nietszeggende Ø en 1 in het commando op te nemen. Het is niet zo moeilijk om dit te doen. Voeg regel 1 bij :

```
1 LET aan = 1 : LET uit = Ø
```

Indien U nu "FLASH aan" schrijft zal het BASIC-systeem de "aan" vervangen door 1 (of "uit" door Ø), het programma wordt hierdoor beter leesbaar.

ARRAYS

Men kan een groep geheugencellen met elkaar in verband brengen door ze dezelfde naam te geven ; men noemt zo een groep een "array" Hoe kunnen we ons een array voorstellen ? Hieronder ziet U een stukje geheugen voorgesteld :



Normaal geven we in BASIC iedere geheugencel een naam (fred, i, enz.) zoals aan de linkerkant wordt getoond. Maar zoals U aan de rechterkant ziet, kunnen we deze vier cellen dezelfde naam geven (we kozen "g"). Bij deze manier van werken moet men echter drie zaken in het oog houden :

1 Een arraynaam bestaat uit slechts één enkele letter, U kunt dus geen array hebben met als naam t2 of albert.

2. Arrays kunnen zo lang zijn als U wenst (natuurlijk binnen het bereik van het geheugen), dus moet U de Spectrum vertellen hoe groot elke array is voor U hem gebruikt. Om dit te doen bestaat er een DIM-statement (afkorting voor dimensie) als volgt :

   ```
   1Ø DIM g(4)
   ```

   voor de array uit ons diagram. Met andere woorden, men geeft de naam van de array en daarachter, tussen haakjes, het aantal cellen dat hij bevat.

3. We moeten bepaalde cellen van de array kunnen identificeren, de techniek om dit te doen wordt in het diagram getoond ; men verwijst naar de eerste cel door g(1), naar de tweede door g(2) enzovoort.

Laat ons even kijken naar de manier waarop men 2Ø getallen in een array kan invoeren. De eenvoudigste manier is :

```
1Ø  DIM n(2Ø)
2Ø  INPUT n(1)
3Ø  INPUT n(2)
4Ø  INPUT n(3)
```

```
.
.
.
210  INPUT n(20)
```

maar U stelt vast dat dit omslachtig is en dat het gebruik van aparte variabelenamen blijkbaar geen voordeel heeft, omdat in de statements 20 tot en met 210 elk array-element wordt genoemd
Wat U moet weten is dat de inhoud tussen haakjes geen getal hoeft te zijn, die kan net zo goed een naam van een variabele zijn. We kunnen dus bijvoorbeeld spreken van n(p) en dat heeft n(2) tot gevolg indien p = 2 en n(17) indien p = 17.
Nu wordt het probleem eenvoudig. We weten dat de waarde tussen de haakjes toeneemt met stappen van 1 vanaf 0 tot en met 20, dat is een kolfje naar de hand van een FOR-lus :

```
10  DIM n(20)
20  FOR p = 1 TO 20
30  INPUT n(p)
40  NEXT p
```

Bij de eerste lusdoorgang is p = 1 en is regel 30 equivalent met INPUT n(1). Bij de tweede doorgang wordt p = 2 en wordt regel 30 INPUT n(2) enzovoort. We hebben de naam van de variabele opzettelijk gekozen als "p" omdat deze letter ook wel "array pointer" wordt genoemd. (Hij "wijst" de hele tijd naar het element uit de array waar naar verwezen wordt.)

MUZIKALE CHRS

We hebben reeds kennis gemaakt met BEEP en we hebben hierbij opgemerkt dat het mogelijk moet zijn om er muziek mee te maken. Zelfs op het meest primitieve niveau zou muziek maken een omslachtige bezigheid zijn omdat we de nummering moeten uitwerken die met elke noot overeenkomt en daarna zouden we een hele reeks BEEP - commando's moeten voorzien van de geschikte getallen en deze invoeren.
Waarom zouden we de computer de noten niet laten vertalen in getallen ?
We zullen voorlopig de zaken eenvoudig houden en in één octaaf werken, namelijk het octaaf waarin zich de grond-C bevindt. De numerieke codes voor deze noten zijn :

A	: −3	D#	: 3
A#	: −2	E	: 4
B	: −1	F	: 5
C	: 0	F#	: 6
C#	: 1	G	: 7
D	: 2	G#	: 8

Omdat de "kruisen" in waarde steeds 1 groter zijn dan de corresponderende noot zonder kruis, hoeven we deze niet te tabuleren, dus is volgende informatie voldoende :

A : −3 , B : −1 , C : 0 , D : 2 , E : 4 , F : 5 , G : 7

Tussen deze getallen bestaat geen eenvoudig verband, dus is het verstandig om ze in een array op te bergen en ze op te zoeken wanneer we ze nodig hebben.
De opslag is eenvoudig :

```
10  DIM s(7)
20  LET s(1) = -3 : LET s(2) = -1 : LET s(3) = 0 : LET s(4) = 2
```

```
30   LET s(5) = 4 : LET s(6) = 5 : LET s(7) = 7
```

Nu voeren we via het toetsenbord een noot (als een letter) in, maar dan moeten we op één of andere manier het geschikte getal in array s kunnen opzoeken. Met andere woorden we hebben iets nodig als :

```
40   INPUT "Voer een noot in" n$
.          [Eén of ander proces dat de juiste waarde van p uit n$
.          genereert (bijv. indien n$ = "A" moet p = 1, indien
.          n$ = "B" moet p = 2, enz.)]
100  BEEP 0.5, s(p)
```

Natuurlijk kunnen we nu een reeks statements gaan schrijven als :

```
45   IF n$ = "A" THEN LET p = 1
50   IF n$ = "B" THEN LET p = 2
.
.
.
```

maar dit is een langdradige manier van werken. We gebruiken beter de interne numerieke codes van de karakters. "A" wordt in werkelijkheid opgeslagen onder het nummer 65, "B" is 66 enzovoort. (Zie appendix A van het *Handboek*.)
U kunt schrijven : LET p = CODE "A". Op die manier krijgt U toegang tot de code voor A. Indien we dus :

```
50   LET p = CODE n$
```

invoeren, dan krijgen we voor p waarden als 65, 66, 67 enzovoort, dus getallen, die 64 te groot zijn. Dus moet regel 50 worden :

```
50   LET p = CODE n$ - 64
```

Nu voegen we een regel toe om de lus terug te doen aansluiten bij het INPUT - statement :

```
110  GO TO 40
```

en nu hebben we een (vrij) primitief klavier. (Denk eraan dat de noten in hoofdletters moeten worden ingevoerd!)
Wat allereerst niet lekker zit is dat we voor de eenvoud de kruisen hebben weggelaten. Indien U "C#" intypt krijgt U C te horen. Omdat de CODE - functie de code neemt van de eerste letter in een string zal de machine de "#" niet hebben opgemerkt!
We moeten dus het tweede symbool in de string onderzoeken, dus n$(2) :

```
60   IF n$(2) = "#" THEN LET i = 1
```

We passen dan regel 100 aan :

```
100  BEEP 0.5, s(p) + i
```

waardoor de gespeelde noot indien i = 1 niet 1 wordt verhoogd. Vanzelfsprekend moeten we bij het begin van de lus i op nul zetten, want anders zal i = 1 blijven wanneer ergens een kruis wordt tegengekomen en zullen alle volgende noten in kruis worden gespeeld :

```
45   LET i = 0
```

Jammer genoeg wordt alles een beetje ingewikkelder door het feit dat n$(2) niet bestaat, indien U een noot-zonder-kruis invoert (zoals B bijvoorbeeld). Tot nu toe werkt het programma feilloos indien U C#, A#, G# invoert, maar wanneer U C

probeert in te voeren krijgt U een foutmelding.
U kunt dit probleempje gemakkelijk oplossen door te definiëren dat n$ een array is met lengte 2 :

```
11   DIM n$(2)
```

Nu bestaat n$(2) altijd. en wanneer het tweede element in de array niet wordt gebruikt zal de Spectrum daar een spatie invullen.
De tweede onvolmaaktheid van ons "instrument" is dat het noten speelt met horten en stoten. Dit is afhankelijk van uw typvaardigheid en van de nukken van het toetsenbord (al bij al is de Spectrum geen elektronisch orgel).
Waarom zouden we de noten niet opslaan in een array in plaats van een noot te laten klinken op het ogenblik dat ze wordt ingevoerd? Op deze manier laten we dan het deuntje spelen als alle noten in de machine zitten. Bijkomend voordeel hiervan is dat we het melodietje kunnen herhalen wanneer we willen.
We hebben dan een andere array nodig om de noten in op te slaan :

```
12   DIM t(1500)
```

Ons melodietje kan dan maximum 1500 noten bevatten.
De invoerlus ondergaat een kleine wijziging omdat het aantal noten niet langer onbeperkt is, en ook omdat de waarde die moet ge-BEEP-t worden in t wordt opgeslagen en niet wordt gespeeld. We moeten ook een uitgang uit de lus voorzien voor het geval er in ons deuntje minder dan 1500 noten voorkomen.
Regels 20 en 30 blijven onveranderd en bevinden zich natuurlijk buiten de lus. We beginnen dus bij regel 35 :

```
35   FOR q = 1 TO 1500
```

Na de input op regel 40 hebben we een test nodig om te zien of de sequentie reeds beëindigd is of niet. We gebruiken "**" om te stoppen :

```
42   IF n$ = "**" THEN GO TO 200
```

en uit de lus te raken.
Regel 100 moet zo worden veranderd dat de gecodeerde noot in t wordt opgeslagen en niet ge-BEEP-t wordt :

```
100   LET t(q) = s(p) + i
```

Aangezien q de opeenvolgende waarden 1, 2, 3 enz. aanneemt, zal de eerste noot worden opgeslagen in t(1), de volgende in t(2) enzovoort.
De lus wordt op regel 110 als volgt gesloten :

```
110   NEXT q
```

inplaats van met GO TO wat tot nu toe het geval was.
Nu kunnen we de melodie vanaf regel 200 laten spelen :

```
200   FOR r = 1 TO q
210   BEEP 0.5, t(r)
220   NEXT r
```

[Opmerking: deze lus loopt q maal, dus niet 1500, omdat bij het verlaten van de eerste lus q het aantal noten van het deuntje bevat.]

Om het deuntje continu te laten herhalen moeten we nog alleen :

```
230   GO TO 200
```

toevoegen
Voor het geval U door de verschillende aanpassingen en bewerkingen in de war mocht gebracht zijn hebben we het volledige programma gelist in de paragraaf over kant - en - klare programma's.
Tenslotte kunnen we een gekleurd display met de muziek laten meespelen. Het een-

voudigste wat we kunnen doen is de kleur van de rand bij iedere gespeelde noot mee laten veranderen. Het zou prettig zijn te schrijven :

```
215  BORDER t(r)
```

zodat elke noot een aparte omlijstingskleur heeft. maar dit kunnen we niet omdat het bereik van de waarden die in t(r) kunnen voorkomen varieert van –3 tot 8 en dit komt niet overeen met het bereik van de mogelijke kleuren dat slechts varieert van 0 tot 7. Indien we bij t(r) 3 optellen ligt het bereik tussen 0 en 11 en indien dit resultaat wordt vermenigvuldigd met 7/11 hebben we het nodige bereik van 0 tot 7. Dus :

```
215  BORDER INT ( (t(r) + 3) * 7/11 )
```

Nu veranderen we de kleur van de achtergrond telkens een noot wordt gespeeld als volgt :

```
216  PAPER 8 * ( r/8 – INT(r/8) ) : CLS
```

Dit vertraagt alles een beetje dus veranderen we 216 in :

```
216  PRINT INK ((t(r)+ 3) * 7/11 ) ; "□□□" ;
```

(Het is niet absoluut noodzakelijk van de uitdrukking te voorzien van INT ; de Spectrum doet dit wel omdat met INK geen andere dan een gehele waarde kan worden geassocieerd.)

16. DEBUGGING III

Bij het debuggen kunnen we ons de vraag stellen "Welke bruikbare gegevens kan de machine ons verschaffen over de manier waarop ze een programma uitvoert?". In dit verband zijn drie punten het overwegen waard :

1. Welke waarden krijgt de machine in de verschillende stadia van de uitvoering voor de variabelen aangeboden?
2. Waar gaat het programma naartoe? (of, anders gezegd, welke regels worden in welke volgorde uitgevoerd?)
3. Hoe dikwijls komt een programma ergens voorbij? (of, anders gezegd, hoeveel keer worden bepaalde regels of groepen van regels uitgevoerd?).

UITPRINTEN VAN DE WAARDEN VAN VARIABELEN

U kunt op een eenvoudige manier waarden van variabelen uitprinten waar U wilt. U moet hiertoe slechts op een geschikte plaats in het programma een PRINT - statement tussenvoegen. Bijvoorbeeld ; in het programma voor de berekening van het gemiddelde van een aantal getallen kunnen we regel 55 inlassen :

```
55  PRINT c
```

waardoor we de veranderingen in de waarde van c tijdens de uitvoering van het programma kunnen volgen. Het is zelfs niet moeilijk om de machine zelf een copie van de dry runtabel te laten uitprinten. Dat is een stuk handiger dan met de hand opstellen. Probeer het maar eens.
Het probleem zit in de keuze van de plaatsen waar de tussenwaarden moeten worden uitgeprint. Dit moet op een verstandige manier gebeuren want anders krijgt U

een sliert getallen waarvan de analyse even lang zou duren als het met de hand opstellen van een dry runtabel.
Bij de tweede run van het "gemiddelden" - programma herinnert U zich nog wel dat op regel 100 een foutrapportering werd uitgeprint en dat we dus geen output kregen van waarden. Een eerste stap wordt gezet door regel 95 toe te voegen :

```
95  PRINT s,c
```

Vergeet niet dat regel 40 moet worden gewijzigd in :

```
40  IF n < 0 THEN GO TO 95
```

Indien U dit laatste niet doet wordt regel 95 nooit uitgevoerd. Door deze aanpassingen zal ons vermoeden worden versterkt dat c nul bevat. Voeg nu regel 65 toe :

```
65  PRINT c ; i ; n ; s ; z
```

waardoor we de volledige reeks variabelen (in alfabetische volgorde) verkrijgen op het einde van elke lus. Hierdoor wordt een vereenvoudigde versie van de dry runtabel verkregen.
Eventueel kunt U gebruik maken van het volgende truukje : tijdens het testen van het programma kan het gebeuren dat U tijdelijk één van de nieuwe regels wilt verwijderen, om te vermijden dat er te veel waarden tegelijk worden uitgeprint. Dit houdt in dat U later de hele regel opnieuw moet intypen, of zelfs – zoals met onze regel 95 – dat U bovendien nog een andere regel moet schrijven omdat regel 40 weer zal veranderd zijn in :

```
40  IF n < 0 THEN GO TO 100
```

indien regel 95 werd verwijderd. Dit is allemaal niet nodig. Het volstaat een REM te schrijven aan het begin van een regel die U niet meer wenst uit te printen. Regel 95 wordt :

```
95  REM PRINT s,c
```

Omdat regel 95 nu een opmerking (REM is afgeleid van REMARK) is geworden, zal de machine op regel 95 geen actie ondernemen, maar een sprong naar regel 95 blijft een geldige actie die niet resulteert in een foutmelding. Wanneer we het statement opnieuw in het programma willen hebben, moeten we slechts het woord REM supprimeren.

SPOORZOEKEN (TRACING)

De eenvoudigste manier om de weg op te sporen die een programma tijdens een run volgt, is na elke regel een PRINT - statement te laten volgen. Hierdoor laat U telkens het regelnummer van de regel die pas werd uitgevoerd, uitprinten. Voor het "gemiddelden" - programma wordt dit bijvoorbeeld :

```
10  LET s=0
11  PRINT "10"
20  LET c=0
21  PRINT "20"
30  INPUT n
31  PRINT "30"
    enzovoort
```

Ook nu weer lopen we de kans dat teveel informatie wordt uitgeprint en dat we door de bomen het bos niet meer zien. Laten we dus in deze "opsporings" - procedure wat selectiever te werk gaan. De vraag waarop een bevredigend antwoord

wordt verwacht, luidt : "maakt het programma een sprong wanneer het de bedoeling is?". Daarom is het verstandig onze "tracing" te beperken tot die programma-onderdelen waarin vertakkingen voorkomen.
Stel bijvoorbeeld dat een routine in een programma een dag van een maand moet invoeren. Zo'n waarde moet liggen binnen het bereik 1 tot en met 31. Het is dus van belang een veiligheid in te bouwen die de gebruiker verhindert een waarde in te voeren die niet in dit bereik ligt. We schrijven hiervoor het volgende stukje programma :

```
 50  INPUT d
 60  IF d > 0 OR d < 32 THEN GO TO 200
 70  REM            Ongeldige input voor d
  .
  .
  .
200  REM            Geldige input voor d
  .
  .
  .
```

Het programma werkt niet zoals het moet. We voegen daarom opsporingsstatements toe na de regels 50, 70 en 200 :

```
 50  INPUT d
 51  PRINT "* 50 *"
 60  IF d > 0 OR d < 32 THEN GO TO 200
 70  REM            Ongeldige input voor d
 71  PRINT "* 70 *"
  .
  .
  .
200  REM            Geldige input voor d
201  PRINT "* 200 *" ;
  .
  .
  .
```

We stellen vast dat het gevolgde spoor voor elke inputwaarde steeds is .

```
* 50 *   * 200 *
```

We gebruiken sterretjes om de spoorregelnummers niet te verwarren met getallen die door het programma worden uitgeprint.
In ons programma zit een fout, want regel 70 wordt blijkbaar nooit aangedaan (het spoor is *steeds* * 50 * * 200 *). We kunnen hieruit één verstandige conclusie trekken : aan de voorwaarde d > 0 OR d < 32 is altijd voldaan. Deze uitspraak is inderdaad steeds waar omdat élk getal groter is dan 0 of kleiner is dan 32 (zelfs 0 en 32 voldoen aan deze voorwaarde omdat 0 kleiner is dan 32 en 32 groter is dan 0).
Regel 60 moet dus worden :

```
60  IF d > 0 AND d < 32 THEN GO TO 200
```

PROGRAMMAPROFIELEN

Een programmaprofiel toont hoeveel keer bepaalde programmaregels werden uitgevoerd.
Stel dat we willen weten hoeveel keer in een bepaald programma regel 420 werd uitgevoerd. We stellen bij het begin van het programma een profielteller (pt) op nul in en vermeerderen deze teller met 1 telkens wanneer regel 420 wordt uitgevoerd:

```
  5  LET pt=0
  .
  .
  .
420  LET a=a*(p-1)
421  LET pt=pt+1
  .
  .
  .
809  PRINT pt
810  STOP
```

Het volgende programma accepteert een reeks van maximaal 20 waarden, die door een nul wordt beëindigd en rangschikt deze waarden in stijgende volgorde. Indien de ingevoerde getallen 3, 8, 1, 4, 2 en 0 zijn, moet het resultaat zijn: 1, 2, 3, 4 en 8.
De nul mag niet in het resultaat verschijnen omdat deze een delimiter is.

```
 10  DIM a(20)
 20  FOR p=1 TO 20
 30  INPUT a(p)
 40  IF a(p)=0 THEN GO TO 60
 50  NEXT p
 60  LET n=p
 65  LET f=0
 70  FOR p=1 TO n
 80  IF a(p) < a(p+1) THEN GO TO 130
 90  LET t=a(p)
100  LET a(p)=a(p+1)
110  LET a(p+1)=t
120  LET f=1
130  NEXT p
140  IF f=1 THEN GO TO 65
150  FOR p=1 TO n
160  PRINT a(p)
170  NEXT p
```

Dit programma werkt niet zoals het hoort. (Typ het in en probeer maar.) Het komt in werkelijkheid in een eindeloze lus terecht.
Waar moeten we beginnen zoeken? De eerste lus (20 - 50) lijkt in orde en de laatste

lus (150 - 170) doet niets anders dan de waarden van array a uitprinten. We concentreren ons dus op de lus van regels 70 tot en met 130. Uit regel 80 blijkt dat soms alle statements van de lus worden uitgevoerd en dat soms de statements op de regels 90 tot en met 120 ongemoeid worden gelaten. We zullen dus twee profieltellers, c1 en c2, nodig hebben die respectievelijk tellen hoeveel keer de lus wordt doorlopen en hoeveel maal het laatste stuk van de lus wordt uitgevoerd. Dit doen we als volgt :

```
 67  LET c1 = 0
 68  LET c2 = 0
 75  LET c1 = c1 + 1
125  LET c2 = c2 + 1
132  PRINT c1, c2
```

We kunnen eveneens na elke lusdoorgang de inhoud van de array uitprinten.

```
134  FOR q = 1 TO n          [ omdat 1 TO n het relevante stuk van
135  PRINT a(q) ;             de array blijkt te zijn ]
136  NEXT q
137  PRINT
```

We proberen een paar sets gegevens en zien wat er gebeurt. Indien we invoeren :

```
3, 6, 1, 8, 5, 0
```

krijgen we als resultaat :

```
6     4
316500
6     4
130056
6     2
100356
6     2
001356
6     1
001356
```

enzovoort tot het geheugen vol raakt.
We stellen vast dat de waarden gerangschikt worden, maar de "8" zijn we kwijt geraakt en waar komen de twee nullen vandaan? Bovendien wordt de hoofdlus constant 6 maal doorlopen maar neemt de frequentie van de doorgang door de sublus gestadig af tot 1 en blijft dan op 1.
Eén nul is duidelijk de delimiter, de andere is een array - element dat niet tijdens de run werd ingesteld maar door het systeem op nul werd geïnitialiseerd. Met andere woorden : het programma behandelt twee waarden te veel. We schrijven regel 60 opnieuw :

```
60  LET n = p - 2
```

en proberen nog eens. Resultaat :

```
4     2
3165
```

```
4     2
1356
4     0
1356
1
3
5
6
```

We maken vooruitgang ; we zijn de nullen kwijt. maar de "8" is nog steeds spoorloos.
Het is moeilijk te zien hoe deze "8" verloren is gegaan. Misschien is ze er toch maar wordt ze niet uitgeprint. Waar printen we ze uit? Regels 150 - 170. Het bereik 1 TO n moet te klein zijn. We vergroten het met 1 :

```
150   FOR p = 1 TO n + 1
```

En natuurlijk zal de opsporing in regel 134 met hetzelfde probleem kampen, dus :

```
134   FOR q = 1 TO n + 1
```

We proberen nogmaals met dezelfde gegevens :

```
4     2
31658
4     2
13568
4     0
13568
1
3
5
6
8
```

Fantastisch, alles is OK. We proberen eens een andere set gegevens :

3, 5, 2, 1, 5, 0

Resultaat :

```
4     3
32155
4     3
21355
4     2
12355
4     1
12355
4     1
12355
enzovoort
```

Het resultaat is goed, maar de lus wordt niet beëindigd. We merken dat c2 nooit nul wordt, in dit geval is de kans groot dat c2 verantwoordelijk is voor het voltooien van het programma.
Welke regel beslist over het al dan niet uitvoeren van de sublus?
Regel 80 :

```
80  IF a(p) < a(p+1) THEN GO TO 130
```

De twee datasets verschillen essentiëel van elkaar in dit punt : de tweede set bevat twee identieke waarden. Omdat 5 niet kleiner is dan 5 blijft het programma de sublus uitvoeren telkens als het paar vijven wordt tegengekomen. Daarom moet regel 80 worden veranderd in :

```
80  IF a(p) <=a(p+1) THEN GO TO 130
```

Nu werkt alles zoals het hoort

```
4     2
32155
4     2
21355
4     1
12355
4     0
12355
1
2
3
5
5
```

17. STRINGS

De postbode brengt U een opvallend persoonlijke brief met de volgende inhoud : "Beste M Peeters, U werd geselecteerd uit een kleine groep mensen die wonen in Bommelskonte om, zonder enige kosten van uwentwege, een prachtig paar varkenslederen Porky laarzen te ontvangen . . ." Erg vleiend natuurlijk, maar Uw buurvrouw, de oude Mevr. Jansen heeft *ongeveer* dezelfde brief ontvangen. En, in werkelijkheid, heeft iedereen uit Bommelskonte en de meeste inwoners van uw streek deze brief ontvangen.
Hier volgt hoe dit wordt gedaan.

```
10  INPUT "Wat is Uw naam?" ; n$
20  INPUT "In welke gemeente woont U?" ; t$
30  INPUT "In welke straat woont U?" ; s$
40  INPUT "Welk huisnummer heeft uw huis?" : h
```

```
50  PRINT n$
60  PRINT h ; "□" ; s$
70  PRINT t$
80  PRINT "Beste □" ; n$ , ", "
90  PRINT "□□□ U werd geselecteerd uit een"
100 PRINT "kleine groep mensen die wonen in"
110 PRINT t$ ; "□ om. zonder enige kosten * van"
120 PRINT "Uwentwege, een prachtig paar"
130 PRINT"varkenslederen Porky laarzen te"
140 PRINT "ontvangen. Wij zijn ervan over-"
150 PRINT "tuigd. □" ; n$ ; "□ dat"
160 PRINT "U van dit uitzonderlijk aanbod"
170 PRINT "zult willen gebruikmaken, en dat"
180 PRINT "de andere inwoners van"
190 PRINT t$ ; "□ groen zullen zien"
200 PRINT "van jaloersheid."
210 PRINT "□□□ Met de minste hoogachting."
220 PRINT "□□□□□ B.E. Drieger"
230 PRINT "□□□□□ Dealer v. Grappen & Grollen"
240 PRINT "* Porto fl 4021.4 extra."
```

Run dit programma en gebruik als INPUT : "Bolle Boos", "Koetjesdorp", "Melkweg" ; "999". Probeer ook andere namen en adressen.
Stel U voor dat dit programma wordt gevoed door een databank en dat het duizenden brieven per uur samenstelt.
Wat ons in deze ridicule oefening opvalt is dat er helemaal geen *computatie* bij te pas komt, het enige wat hier gebeurt is geheugenopslag en erg eenvoudige tekstmanipulatie. De computer kan geschreven teksten even goed aan als getallen omdat hij *strings* kan opslaan. De dollarsymbolen ($) stellen strings voor.
Een string is een sequentie van *karakters*. De karakters worden in het *handboek* gelist en elk karakter heeft een CODE waarover we het zo dadelijk zullen hebben.
Hier volgt een string :

stringstringstringstringstring

En hier volgt een andere string :

a b 3 3 4 * . / > > > > < < – b + + + + + q q j.

(Indien U de punten typt maken die ook deel uit van de string)
Een string kan ook één karakter lang zijn, zoals <, en kan zelfs helemaal *geen* karakters bevatten!
Om een string toe te kennen aan een stringvariabele moet men de string voorzien van aanhalingstekens :

```
10  LET a$ = "stringstringstringstringstring"
```

Iedere stringvariabele moet een enkelvoudige letter zijn waarop het $-teken volgt.
Voor een string met lengte nul is LET a$ = "" voldoende.
U kunt strings manipuleren. Beschouw bijvoorbeeld een string met één karakter, zoals :

3

Dit kan worden geïnterpreteerd als :

(a) een getal, 3
(b) een string, "3"

en U kunt op verschillende manieren van de éne interpretatie naar de andere overschakelen. Stel dat we het als een string beschouwen en toekennen :

```
10 LET a$ = "3"
```

Stel dat U 3 + 5 wilt uitwerken. Dan is :

```
20 LET b = a$ + 5
30 PRINT b
```

zinloos. Waarom? We hebben de domme computer opgedragen 3 te beschouwen als een string en hij weet niet dat 3 ook een *getal* is. Maar we kunnen d.m.v. VAL een string *omzetten* in een getal :

```
20 LET b = VAL a$ + 5
30 PRINT b
```

Algemeen : indien we een string hebben die toevallig een rekenkundige uitdrukking is dan weet de machine niet dat ze kan worden uitgewerkt. Bijvoorbeeld

```
10 LET a$ = "2 + 2 + 5 * 3"
```

beschouwt de machine als een sequentie van *karakters* :

2 + 2 + 5 * 3

U kunt bijvoorbeeld het zesde karakter uit een string halen met :

```
20 PRINT a$(6)
```

en als antwoord * krijgen. Als rekenkundige uitdrukking is het resultaat natuurlijk 19 en dit *heeft* geen zesde karakter. Maar indien U de string wilt omzetten in een getal, dan zal :

```
20 PRINT VAL a$
```

19 als antwoord geven.
U kunt op twee manieren een getal omzetten in een string. De eerste manier plaatst aanhalingstekens rond het getal. Maar indien U in een programma bijvoorbeeld a + b uitwerkt, waarvan het resultaat 3336 is, moet U niet "a + b" schrijven en hopen dat "3336" in het programma wordt opgenomen; wat U dan wel krijgt is een driekarakterstring, a + b, en dat was niet de bedoeling.
CHR$ zet een getal tussen 0 en 255 om in één enkel karakter (en dit overeenkomstig de codelist uit het *handboek*). Bijvoorbeeld, CHR$96 is het pond sterlingsymbool £. Bepaalde getallen worden niet gebruikt.
CODE doet het omgekeerde : CODE "£" is 96. Indien U CODE "£ 335/h" probeert, krijgt U nog steeds 96 omdat deze functie slechts het eerste karakter bekijkt.
LEN vertelt U hoe lang een string is.
Veruit de meest interessante eigenschap met strings is deze waarmee U stukjes uit string kunt halen. Deze stringfragmentjes noemen we *substrings*. De instructie :

```
a$(3 TO 7)
```

levert een string op die op zijn beurt bestaat uit het derde, vierde, vijfde, zesde en zevende karakter van a$. Indien a$ = "stringstringstringstringstring" dan is a$(3 TO 7) = "rings". In de plaats van 3 en 7 kunt U willekeurige getallen gebruiken. De instructie a$(5) pikt het vijfde karakter – hier "n" – uit a$.
Stel bijvoorbeeld dat U een getal tussen 1 en 7 wilt invoeren en dat de machine dan

moet antwoorden welke dag dat is (waarbij 1 zondag is, 2 maandag, enzovoort). Een onhandige manier om dit te doen is:

```
10  INPUT n
20  IF n = "1" THEN PRINT "Zon"
30  IF n = "2" THEN PRINT "Maa"
  .
  .
  .
```

totdat de zeven dagen werden genoteerd.
Indien U substrings gebruikt heeft U vijf regels minder nodig:

```
10  LET a$ = "ZonMaaDinWoeDonVryZat"
20  INPUT n
30  PRINT a$(3 * n – 2 TO 3 * n)
```

Indien het systeem niet helemaal duidelijk is, zie dan onder DAGZOEKER bij de kant-en-klaar programma's in dit boek.
U kunt ook de staart van een string verbinden met de kop van een tweede string d.m.v. +:

```
"hot" + "dog" = "hotdog"
```

of U kunt ze "alfabetisch" rangschikken door < te gebruiken. (Raadpleeg het *handboek* hiervoor en experimenteer.) Door strings zorgvuldig te gebruiken kan men dikwijls veel ruimte sparen.
Dit programma zoekt aan de hand van een naam, de initialen.

```
10  INPUT "Wat is Uw volledige naam?" ; n$
20  LET n$ = "□" + n$
30  FOR i = 1 TO LEN n$
40  IF n$(i) = "□" AND i < LEN n$ THEN PRINT n$(i + 1) ; "." ;
50  NEXT i
```

Run dit programma en gebruik uw naam. Voer "Ginormous Electronic Corporation" in en kijk of "G.E.C." wordt uitgevoerd.
In deze vorm is het programma niet perfect. Indien U *extra* spaties tussen de woorden plaatst wordt de uitvoer nogal slordig. Dit programma doet niets anders dan een extra spatie aan de kop van een string hangen en elk karakter dat *volgt* op een spatie als een initiaal uitprinten. Voor de duidelijkheid worden puntjes tussengevoegd.

PROJECT

Pas het programma zo aan dat het herhaalde spaties negeert. Dit kan bijvoorbeeld door de string af te tasten en elke spatie die voorkomt na een bepaalde spatie te schrappen. Om het j-de karakter in een string n$ te schrappen schrijft U:

```
LET n$ = n$(TO j – 1) + n$(j + 1 TO)
```

Indien U de getallen voor of na de "TO" weglaat, neemt de machine aan dat dit respectievelijk kop en staart van de string zijn.

18. GEGEVENS

Boven toets D vindt U, in het groen, het woord "DATA". Indien U deze instructie gebruikt, hoeft U geen lange reeksen statements van het type "LET a(37)=242, LET a(38)=243, enzovoort" in te typen. Bij wijze van voorbeeld en om U met het commando vertrouwd te maken typt U hetvolgende in :

```
10  DATA 1, 2, 3, 4, 5, 6, 7
20  FOR i = 1 TO 7
30  READ x
40  PRINT x
50  NEXT i
```

In de uitvoer moet U opnieuw de getallen 1 tot en met 7 krijgen. Het READ - commando leest U als "LET x = het volgende getal van de DATA - list". De computer doorloopt de DATA - regels telkens als een READ - instructie wordt tegengekomen, zoekt dan welk item het laatst werd gelezen en vult tenslotte het volgende item in als waarde voor de variabele van het READ - statement. Bij de eerste doorgang leest hij voor x de waarde 1, de tweede keer 2, enzovoort.
DATA is een erg nuttige instructie omdat de machine alle DATA - regels in een programma beschouwt als één enkele list (het einde van de eerste wordt vastgemaakt aan het begin van de tweede list enzovoort). Deze regels moogt U plaatsen waar U wilt (maar. het is wel sneller wanneer U ze zoveel mogelijk vooraan plaatst omdat de computer ze systematisch aftast). Schrijf het programma opnieuw als volgt :

```
10  FOR i = 1 TO 7
20  READ x
30  DATA 1, 2, 3, 4
40  PRINT x
50  DATA 5
60  NEXT i
70  DATA 6, 7
```

U stelt vast dat, zelfs indien de DATA - regel zich in een FOR/NEXT - lus bevindt, het resultaat hetzelfde is. DATA vormt dus een uitzondering op de stelregel die zegt dat regels in numerieke volgorde worden uitgevoerd.
Indien dit alles is wat we met DATA kunnen doen, dan zou het niet erg opwindend zijn. Maar, er is meer.

```
10  DATA 100, 50, 150, 100, 100, 150, 50, 100,100, 50
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 4
50  LET x0 = x : LET y0 = y
60  READ x, y
70  DRAW x – x0 , y – y0
80  NEXT i
```

Dit programma tekent een ruitvormige figuur: de DATA - list geeft de coördinaten

van de hoeken waarbij de onderste hoek wordt herhaald als aankomst- en startpositie.
Laten we deze idee uitwerken in een echt "kattig" programma :

```
10  DATA 2, 0, 6, 0. 9, 1, 15. 0, 16, 1, 16, 12, 15, 13, 14. 12, 14. 2,
    10, 2, 11, 6, 8, 12, 10, 15. 9, 18, 8, 22, 7, 18, 3, 18, 2, 22, 1, 18. 0,
    15, 2, 12, 0, 6, 0, 3. 2, 0
20  READ x, y
30  PLOT x, y
40  FOR i = 1 TO 23
50  LET x0 = x : LET y0 = y
60  READ x, y
70  DRAW x – x0, y – y0
80  NEXT i
```

U stelt vast dat eerst het punt met coördinaten 2,0 wordt geplot en dat daarna dit punt wordt verbonden met punt 6,0 ; daarna met punt 9,1 enz. De getallen volgen elkaar op in de DATA - list.
Om de figuur meer naar het midden van het scherm te plaatsen kunt U regel 30 veranderen in :

```
30  PLOT 100 + x, 60 + y
```

De DATA - list werd samengesteld door een ruwe schets op millimeterpapier te tekenen, waarvan de coördinaten van de punten werden afgelezen. De werkwijze is dus dezelfde als die in punt-tot-punt tekenboekjes wordt gebruikt, maar wij moeten d.m.v. coördinaten de punten aanduiden.
Om een kat te tekenen is dit toch een boel werk. Indien U dikke katten, magere katten, katten op hun kop en katten die rug aan rug zitten, wenst te tekenen kunt U de gegevens *transformeren*. Pas regel 30 aan opdat U voldoende plaats zult hebben en voeg deze regel bij :

```
5  INPUT a, b
```

en verander regel 70 in :

```
70  DRAW a * (x – x0), b * (y – y0)
```

Nu moet U bij de RUN twee getallen invoeren. Wees in het begin niet te overmoedig en start met a = 1, b = 2 ; en a = 2. b = 1. Probeer dan a = 2, b = –1 ; a = –2, b = –1.
Probeer nu naar hartelust maar denk eraan dat in het programma geen bescherming werd ingebouwd tegen het van het scherm lopen. Bovendien geven gebroken waarden voor a en b gekke resultaten die te wijten zijn aan afrondingsfouten. Indien U een truukje wilt gebruiken om dit te vermijden, zie dan onder SPIRALEN bij de kant-en-klaar-programma's.
Laten we nu eens een hele regel met katten tekenen. Dit kunnen we met behulp van een lus ; maar we hebben dan een middeltje nodig om de READ - instructie naar het begin van de DATA - list te "resetten". De RESTORE - instructie maakt dit mogelijk, dus voegen we volgende regels toe :

```
15   FOR t = 1 TO 8
90   RESTORE
100  NEXT t
```

en we wijzigen regel 30 als volgt :

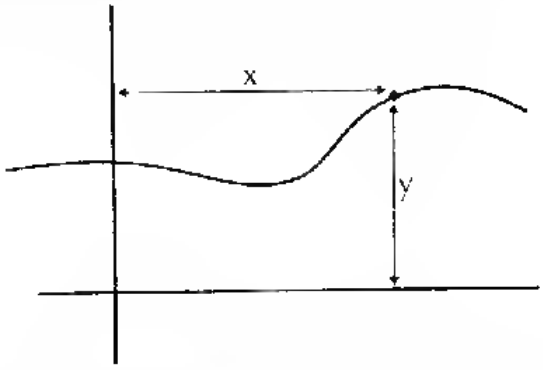
```
30  PLOT 50 + x + 20 * t, 50 + y
```

PROJECTEN

1. Laat de katten, naarmate ze meer naar rechts worden getekend, een trapje hoger zitten.
2. Voeg gegevens toe zodat ook de treden van de trap worden getekend.
3. Wijzig met behulp van een atlas de DATA - list zó dat het programma een kaart van Australië tekent. Kijk eens hoe Australië er op zijn kop uitziet (de Australiërs denken immers dat Australië op deze manier moet worden bekeken).
4. Indien U een halve middag de tijd heeft, kunt U een DATA - list opstellen om een wereldkaart mee te tekenen.

19. KROMMEN UITPLOTTEN

Waarschijnlijk hebt U al wel eens gehoord van een *grafiek* die wordt gespecificeerd d.m.v. coördinaten.
We beginnen met twee rechten die loodrecht op elkaar staan, de x - as en de y - as. Met behulp van deze assen kunnen we afstanden x en y afpassen (negatieve waarden voor x, vinden we links van de y - as en negatieve y - waarden onder de x - as). Deze afstanden kunnen we gebruiken om een punt te specificeren met *coördinaten* x en y, net als bij pixels (figuur 19.1)



Figuur 19.1 Coördinaten voor uitplotting van krommen

Stel U nu voor dat x variëert volgens de x - as en dat y variëert op een manier die afhankelijk is van x, dan zal het punt met coördinaten (x,y) ook bewegen, en meestal beschrijft het een *curve* (of *kromme*). Indien U in een formule aangeeft hoe y van x afhankelijk is (bijvoorbeeld $y = x^2 - 3$) dan hebt U meteen een formule voor die kromme.
We kunnen dergelijke krommen tekenen met behulp van PLOT - instructies :

```
10  FOR j=0 TO 255
20  PLOT j,j/2
30  NEXT j
```

Hierdoor krijgt U een spoor van pixels dat van de linker benedenhoek van het scherm naar de rechterbovenhoek klimt. Een wiskundige zou dit de grafiek van de functie y = x/2 noemen.

Door deze functie te wijzigen krijgt U een onuitputtelijke variëteit van grafieken. Het is voldoende dat U j/2 vervangt door een andere uitdrukking waarin j voorkomt. Indien U bijvoorbeeld de vierkantswortel van j als functie van j wil uitplotten kunt U j/2 vervangen door SQR j :

```
10  FOR j=0 TO 255
20  PLOT j, SQR j
30  NEXT j
```

Experimenteer door regel 20 als volgt te veranderen :

```
(Parabool)     20  PLOT j, .002 * j * j
(Sinusoïde)    20  PLOT j, 80 * SIN(j/20) + 80
(Cosinusoïde)  20  PLOT j, 80 * COS(j/20) + 80
(Kettinglijn)  20  PLOT j, (EXP(.02 * (j - 120) + EXP(.02 * (120 - j))) * 10
```

De uitdrukkingen in j zijn nogal ingewikkeld omdat anders de figuren niet op het scherm passen. De waarde van de vertikale coördinaat moet liggen tussen 0 en 175. Om alles te doen passen moet U de *schaal* aanpassen. Door hetvolgende eens te proberen zult U vlug inzien waarom dit nodig is :

```
20  PLOT j, j * j
20  PLOT j, SINj
20  PLOT j, COSj
20  PLOT j, EXP(j) + EXP(-j)
```

Natuurlijk bestaan er manieren waarmee U dit probleem kunt omzeilen (zie verder onder SCHAAL.) Maar probeer eerst de volgende functies die wel op het scherm passen omdat we ze zorgvuldig hebben uitgekozen :

```
20  PLOT j, EXP(-j/80) * SIN(j/8) * 80 + 80
20  PLOT j, 80 + 80 * COS SQR(2 * j)
20  PLOT j, ABS (j - 127)
20  PLOT j, 80 + 60 * LN(1 + ABS SIN(j * .125))
20  PLOT j, 12 * ABS(j/4 - 30) ↑ .666
20  PLOT j, 40 * ABS(j/4 - 30) ↑ .25
20  PLOT j, 160 * EXP(-.1 * (j/4 - 30) * (j/4 - 30))
```

Om het schrijven van ingewikkelde uitdrukkingen te vermijden kunt U de formule in trapjes opbouwen zoals in :

```
20  LET t=j/24
25  PLOT j, 120 + .1 * t * (t - 2) * (t - 4) * (t - 6) * (t - 8) * (t - 10)
```

SCHAAL

Voorlopig werken we slechts met functies die gedefinëerd zijn voor *positieve* getallen en die positieve waarden opleveren. Een goed voorbeeld is SQR, de vierkantswortel. Neem bovenstaand programma voor het uitplotten van een grafiek en vervang regel 20 achtereenvolgens door :

```
(a)  20  PLOT j, SQR j
(b)  20  PLOT j, 2 * SQR j
(c)  20  PLOT j, 4 * SQR j
```

(d) 20 PLOT j, 6 * SQR j

(e) 20 PLOT j, 8 * SQR j

(f) 20 PLOT j, 10 * SQR j

(g) 20 PLOT j, 12 * SQR j

(h) 20 PLOT j, 14 * SQR j

U merkt vrij vlug dat in de opeenvolgende grafieken alles hoger op het scherm komt en dat in (g) de machine stilvalt omdat punten van de grafiek van het scherm beginnen af te lopen. Hoe groter de "factor" in (factor) * SQR j wordt, hoe meer de grafiek in de vertikale richting wordt uitgerekt. Deze "factor" is een *schaalfactor*. Een goede keuze van de schaalfactor maakt het mogelijk dat een grafiek netjes op het scherm past.
Indien de schaalfactor te klein wordt genomen, krijgt U grafieken die zo sterk zijn platgedrukt dat U niets kunt zien. Probeer maar eens met:

```
20 PLOT j, .1 * SQR j
```

Indien de functie die U aan het uitplotten bent te groot wordt, kunt U ze terug op het scherm krijgen door de schaalfactor kleiner te maken. Bijvoorbeeld:

```
20 PLOT j, j * j
```

loopt van het scherm af omdat 14 * 14 = 196 en dat is reeds te groot. In feite is het grootste getal dat U moet uitplotten 255 * 255 = 65025. Indien U dit getal deelt door 400 krijgt U 162.5625 en dit past nog, omdat 175 nog net toegelaten is. U krijgt dus een mooie grafiek op voorwaarde dat U als schaalfactor 1/400 gebruikt:

```
20 PLOT j, j * j/400
```

Er bestaat een algemene regel om de schaalfactor op een geschikte manier te kiezen. Stel dat de grootste waarde die de functie aanneemt indien j varieert van 0 tot 255, m is (m staat voor maximum). Dan wordt met schaalfactor s alles mooi geschikt op voorwaarde dat s * m niet groter is dan 175, en liefst deze 175 zo dicht mogelijk benadert.
U kunt s * m gelijk maken aan 175 door voor s de waarde s = 175/m te nemen. Voor ronde figuren is 160/m beter en in dit geval heeft U meer aan een redelijke schatting voor m dan aan een exact getal.
U kunt zelfs een programma schrijven om m uit te werken. Stel dat we bij de j * j functie blijven, dan doet hetvolgende programmaatje het werk voor U:

```
10 LET m = 0
20 FOR j = 0 TO 255
30 LET q = j * j
40 IF q > m THEN LET m = q
50 NEXT j
```

Indien U de uitplotroutine hieraan vastplakt, zal de grafiek in zijn ideale vorm worden uitgetekend:

```
60 FOR j = 0 TO 255
70 PLOT j, (175/m) * j * j
80 NEXT j
```

Er is slechts één gebrekje: U moet alle berekeningen *tweemaal* doen. Dit is moeilijk op een efficiënte wijze te vermijden tenzij U weet *welke* j de grootste waarde voor j * j geeft. In dit geval is het duidelijk dat dit j = 255 is, maar zo eenvoudig is het niet steeds. (U *kunt* eventueel een vector v(i) met grootte 256 dimensioneren, de waar-

den van j * j als v(j + 1) opslaan en deze laatste waarde voor de uitplotting gebruiken; maar vectoren en arrays nemen veel geheugenruimte in beslag! Voor verdere informatie over vectoren en arrays raadpleegt U best het *handboek.*)
U kunt de schaal van de horizontale as ongeveer op dezelfde manier aanpassen als we dit met de schaal van de vertikale as hebben gedaan. De functies SQR j of j * j tonen dit niet duidelijk, dus gebruiken we de functie 80 + 80 * SIN j waarin dit wel tot uiting komt.
Probeer volgende functies eens:

```
20  PLOT j, 80 + 80 * SIN(.05 * j)
20  PLOT j, 80 + 80 * SIN(.1 * j)
20  PLOT j, 80 + 80 * SIN(.15 * j)
20  PLOT j, 80 + 80 * SIN(.2 * j)
20  PLOT j, 80 + 80 * SIN(.25 * j)
```

Nu verandert de *horizontale* schaal, maar de schaalfactor gedraagt zich op een andere manier: hoe groter de schaalfactor wordt, hoe meer golfjes de grafiek gaat vertonen.
Wanneer j variëert tussen 0 en 255, variëert het getal .05 * j tussen .05 * 0 en .05 * 255 of, anders uitgedrukt, tussen 0 en 12.75
Wanneer j varieert tussen 0 en 255, varieert .1 * j tussen 0 en 25.5.
Dus worden in het tweede geval *tweemaal* zoveel waarden in dezelfde horizontale richting samengedrukt.
In werkelijkheid wordt met een schaalfactor s, dus als in:

```
20  PLOT j, 80 * 80 * SIN(s * j)
```

het bereik van 0 tot s * 255 over de breedte van het scherm uitgeplot. Hoe groter s wordt, hoe groter ook het bereik wordt en hoe meer alles in horizontale richting wordt samengedrukt.
Indien U dus over een gekozen bereik wenst uit te plotten, bijvoorbeeld 1000, dan moet s * 255 = 1000, en dat is hetzelfde als zeggen dat s = 1000/255. Indien U het bereik van 0 tot n wenst uit te plotten, dan heeft U een schaalfactor n/255 nodig.
Samengevat:

$$\text{beste vertikale schaalfactor} = \frac{175}{\text{grootste uitgeplotte waarde}}$$

$$\text{beste horizontale schaalfactor} = \frac{\text{bovengrens van het bereik van de variabele}}{255}$$

PROJECT

Indien U niet goed weet wat het mooiste resultaat zal geven, dan kunt U een "interactief" programma schrijven dat U (via INPUT) de twee schaalfactoren laat kiezen en daarna de grafiek uitplot. Indien het resultaat U niet bevalt, probeert U een volgende run en kiest U andere schaalfactoren.
Schrijf zo een programma voor de functie 80 + 80 * SIN j.
Tip: indien h de horizontale en v de vertikale schaalfactor is dan luidt regel 20:

```
20  PLOT j, v * (80 + 80 * SIN(h * j))
```

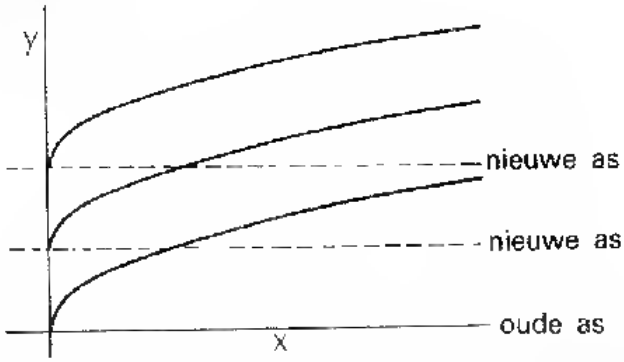
ASVERSCHUIVINGEN

Misschien vraagt U zich af waarvoor al die "80 + 80 * SIN j" - uitdrukkingen nodig zijn of wat er gebeurt indien de getallen negatief zijn. In beide gevallen is het ant-

woord hetzelfde. men kan slechts met negatieve getallen werken door de assen te verschuiven.
Probeer dit programma :

```
10  INPUT s
20  FOR j=0 TO 255
30  PLOT j, s+SQRj
40  NEXT j
```

Voer voor s de waarden 0, 10, 20 enzovoort in
U ziet dezelfde grafiek die, naargelang de waarde van s, op het scherm omhoog wordt verschoven. Men kan deze manipulatie op twee manieren beoordelen.
Een eerste interpretatie is dat men op deze manier verschillende functies uitplot zoals 10 + SQR j of 20 + SQR j.
Een tweede manier van interpreteren is dat men steeds dezelfde functie, namelijk SQR j, uitplot maar dat de positie van de x - as op het scherm verandert. Figuur 19.2 spreekt voor zichzelf.



Figuur 19.2 Een constante optellen komt neer op het verplaatsen van de x - as

Om bijvoorbeeld een duidelijke sinuscurve te krijgen, moet men bovenstaande SQR j - functie vervangen door de functie 80 * SIN(.1 * j) en s zo nemen dat alles centraal op het scherm komt. s = 80 werkt prachtig, vandaar al deze 80 + 80 * SIN - uitdrukkingen.
Op deze manier wordt de x - as verschoven. Om de y - as te verplaatsen kunt U het bereik van j aanpassen, bijvoorbeeld van –120 tot 135 inplaats van 0 tot 255.
U kunt deze verschuivingen combineren met schaalaanpassingen in beide richtingen; U hoeft de assen niet centraal te houden. We stellen dus voor dat U het best kunt experimenteren totdat U een bevredigend resultaat behaald hebt. Om U bij dit experimenteren te helpen kunt U volgend programma gebruiken :

```
10   INPUT a, b, c, d
20   PLOT 0,d
30   DRAW 255,0
40   LET u=-b/a
50   PLOT u,0
60   DRAW 0,175
100  FOR j=0 TO 255
110  PLOT j,c*SIN(a*j+b)+d
120  NEXT j
```

In dit programma verschuiven a, b, c en d de assen en passen ze de schaal aan en worden x - as en y - as getekend. We hebben dit programma niet beschermd tegen vastlopen indien de functie buiten het bereik komt: hierdoor zal men zorgvuldig de schaal en de asposities kiezen.
(Probeer a = .1, b = −10, c = 80, d = 80. Dikwijls is een negatieve waarde voor b noodzakelijk)

20. DEBUGGING IV

Tot nu toe hebben we slechts gekeken naar fouten die we zelf hadden veroorzaakt en die vrij gemakkelijk konden worden hersteld zodra we ze hadden opgemerkt. Maar er bestaat een andere soort fouten. Het betreft hier geen ontwerpfout maar het heeft te maken met de nauwkeurigheid waarmee computers getallen opslaan. Als we nadenken over de gewone manieren waarop men getallen kan opslaan, is het vanzelfsprekend dat er grenzen zijn aan het aantal cijfers die kunnen worden onthouden. De kilometerteller van een auto kan slechts vijf of zes cijfers bevatten omdat hij slechts vijf of zes "wieltjes" heeft. Met een computer is het net zo. Elk getal kan niet meer dan een vast aantal "wieltjes" beslaan. Hoewel bij computers elk "wieltje" geen decimaal cijferteken voorstelt. De interne machinecode voor getallen is totaal verschillend van de wijze waarop wij over getallen denken, maar we zullen U niet lastig vallen met alle details hieromtrent. Het feit dat een inherente onnauwkeurigheid *en* een codeconversie aanwezig zijn, brengen met zich dat de externe voorstelling van een getal niet dezelfde is als de interne voorstelling van datzelfde getal. We geven een voorbeeld aan de hand van logaritmen. Indien U, gebruikmakend van logaritmen, 2 met 2 vermenigvuldigt krijgt U :

Getal	Logaritme
2	0.3010
2	0.3010
3.999	0.6020 +

d.i. 2 × 2 = 3.999 !
De onnauwkeurigheid ontstaat door de combinatie van, enerzijds een codeconversie die wordt verricht (getal naar logaritme en van logaritme terug naar getal) en anderzijds door het feit dat logaritmen slechts nauwkeurig zijn tot op vier cijfers na de komma (anders gezegd: ze kunnen slechts vier "wieltjes" beslaan).
Volgend programma stelt een gelijksoortig probleem voor :

```
10  FOR p=0 TO .2 STEP .01
20  LET q=ATN(TAN(p))
30  IF p < > q THEN PRINT p,q
40  NEXT p
```

Op regel 20 nemen we de tangens van p en onmiddellijk daarna wordt het proces omgekeerd door de arcustangens van het resultaat te nemen. Met andere woorden: q moet gelijk zijn aan p. Dus zou regel 30 nooit mogen resulteren in het uitprinten van p en q (want deze waarden zijn steeds gelijk.) Wanneer we het programma runnen, levert dit dus volgend resultaat op:

```
0.02      0.02
0.03      0.03
0.04      0.04
0.05      0.05
0.07      0.07
0.09      0.09
0.11      0.11
0.12      0.12
0.13      0.13
0.14      0.14
0.16      0.16
0.18      0.18
0.2       0.2
```

Dit is inderdaad een eigenaardig resultaat dat te wijten is aan het feit dat de machine niet alleen waarden uitprint en dus beweert dat deze waarden verschillend zijn, maar op de koop toe worden de waarden uitgeprint alsof ze identiek zijn! In werkelijkheid heeft het ingewikkeld wiskundig proces dat hierbij betrokken is aanleiding gegeven tot zeer kleine onnauwkeurigheden in de interne voorstellingswijze van de getallen. Het zijn deze onnauwkeurigheden die leidden tot de verschillen tussen p en q. Nochtans zijn er ook onvolkomenheden tengevolge van het opnieuw decoderen van het interne format naar de decimale getallen die op het scherm verschijnen waardoor ze als identiek tevoorschijn komen hoewel de machine onvermurwbaar volhoudt dat ze dit niet zijn. Merk op dat voor bepaalde waarden de interne codes identiek *zijn*: bijvoorbeeld voor 0.06 en 0.08.
In plaats van getalwaarden op gelijkheid te onderzoeken, bestaat de enige uitweg in het toelaten van een kleine afwijking in het IF - statement, zodat we krijgen:

```
IF ABS(p-q) < 0.000001 THEN ...
```

De ABS - functie is noodzakelijk omdat q groter kan zijn dan p. Indien bijvoorbeeld p = 3 en q = 3.1 dan is p − q = −0.1 en dit is kleiner dan 0.000001, dus wordt aan de voorwaarde voldaan indien de ABS - functie (die op een effectieve manier het minteken uit de uitdrukking wipt) er niet zou staan ABS(−0.1) = 0.1 en dit is groter dan 0.000001, dus wordt niet aan de voorwaarde voldaan en dit was onze bedoeling.

21. PROGRAMMEERSTIJL

We beginnen met een programma dat U eventueel kunt beschouwen als een kant-en-klaar-programma; copieer en run het.

Er worden eerst twee steden en twee anti-raketsilo's afgebeeld. De silo's worden van links naar rechts aangeduid als "1" en "2". Vervolgens komt er een regen van raketten naar beneden op een manier die niet te voorspellen is, maar de baan die een bepaalde raket beschrijft heeft een vaste hoek.
U drukt op toets "1" of op "2" van het toetsenbord om één van beide verdedigingssilo's te activeren. (Het kan even duren voor het programma hierop reageert, houd dus de toets zolang ingedrukt.) Nu wordt U verzocht een vliegbereik en een hoek op te geven voor de afweerraketten. Indien U een neerkomende raket raakt, houdt de kop van het vijandig projektiel op met flitsen.
Op een bepaald ogenblik zult U geen afweerraketten meer hebben (bij het begin beschikt U in iedere silo over 20 raketten) of zullen uw silo's of steden vernietigd zijn. Het komt erop aan zoveel mogelijk vijandige projektielen neergehaald te hebben voor dit gebeurt.
We verklaren eerst nader hoe U de waarden van het "vliegbereik" en van de "schiethoek" moet invoeren.
Het vliegbereik wordt ingevoerd via het coördinatensysteem dat de Spectrum gebruikt voor de hoge resolutie-grafieken (mode PLOT). Indien U dus een object wilt treffen dat zich vlak boven een silo bevindt, dan is het bereik 175. De afstand tussen de linker benedenhoek en de rechter bovenhoek bedraagt ongeveer 300.

De schiethoek (gemeten in graden) wordt gevormd door de horizontale as en de rechterkant van een silo. Dit heeft tot gevolg dat de schiethoeken van silo 2 tussen 90° en 180° liggen. wees dus voorzichtig! (Voor silo 1 beschikt U over een schietveld dat ligt tussen 0° en 90°!) Een eigenschap van het programma is dat het U niet vertelt of U een fout maakt. Indien U bijvoorbeeld een raket afvuurt naar een doel dat buiten het scherm ligt dan zal het programma geen baan uittekenen maar wel een raket uit uw voorraad weghalen.
Hier volgt de volledige listing van het programma dat we de naam "Anti - missile screen" hebben gegeven; de verklaring van de betekenis van een aantal gebruikte variabelenamen wordt gegeven bij de analyse van het programma.

ANTI - MISSILE SCREEN

```
1     RANDOMIZE
10    LET display = 500: LET genmis = 1000
20    LET movmis = 1500
30    LET fire = 2500: LET prnscr = 3000: LET hitest = 3500
40    DIM m(3,20): DIM s(2)
50    LET city = 2: LET s(1) = 20: LET s(2) = 20
60    LET kilsilo = 4000. LET kilcity = 4500
70    LET score = 0
80    LET kbhit = 5000
100   GO SUB display
110   IF city < 1 OR s(1) + s(2) = 0 THEN GO SUB prnscr
120   GO SUB genmis
130   GO SUB movmis
160   GO SUB 110
500   LET xs = 8
510   FOR l = 1 TO 2
520   FOR x = xs TO xs + 20
530   PLOT x,0
540   DRAW 0,4
550   NEXT x
560   LEXT xs = 224
570   NEXT l
580   LET xs = 56
590   FOR l = 1 TO 2
600   FOR x = xs TO xs + 36 STEP 3
610   LET h = INT(RND * 10) + 3
620   FOR y = 0 TO h
630   PLOT x,y
640   DRAW 3,y
650   NEXT y
660   NEXT x
670   LET xs = 144
680   NEXT l
690   RETURN
1000  FOR l = 1 TO 5
1010  LET x = INT(RND * 100)
1020  LET a = (RND * PI/2) + .01
1030  FOR p = 1 TO 20
1040  IF m(3,p) = 0 THEN LET m(1,p) = x:
```

```
     LET m(2,p)=175. LET m(3,p)=a: GO TO 1060
1050 NEXT p
1055 GO SUB kbhit
1060 NEXT l
1070 GO SUB kbhit
1080 RETURN
1500 FOR p=1 TO 20
1505 IF m(3,p)=0 THEN GO TO 1580
1510 PLOT FLASH 0 ; m(1,p). m(2,p)
1520 LET xo=20 * COSm(3,p)
1530 LET yo=-20 * SINm(3,p)
1540 IF m(1,p)+xo > 255 OR m(2,p)+yo < 0
     THEN LET m(3,p)=0 : GO SUB hitest : GO TO 1580
1550 DRAW xo, yo
1560 LET m(1,p)=m(1,p)+xo
1570 LET m(2,p)=m(2.p)+yo
1573 PLOT FLASH 1 ; m(1,p), m(2,p)
1575 GO SUB kbhit
1580 NEXT p
1585 GO SUB kbhit
1590 RETURN
2500 INPUT "bereik, hoek :" : rg, ag
2505 IF s(VAL dS)=0 THEN RETURN
2510 If d$="1" THEN LET xb=18
2520 IF d$="2" THEN LET xb=234
2523 PLOT xb,0
2525 LET s(VAL d$)=s(VAL d$)-1
2530 LET xf=rg * COS(ag * PI/180)
2540 LET yf=rg * SIN(ag * PI/180)
2545 IF xf+xb < 0 OR xf+xb > 255 OR yf < 0 OR yf > 175
     THEN RETURN
2550 DRAW xf,yf
2560 FOR p=1 TO 20
2570 IF ABS(xf+xb-m(1,p)) > 15 OR ABS(yf-m(2,p)) > 15
     THEN GO TO 2700
2610 LET score=score+1
2625 PLOT FLASH 0 : m(1,p), m(2,p)
2630 LET m(3,p)=0
2700 NEXT p
```

```
2710  RETURN
3000  CLS
3010  IF city < 1 THEN PRINT AT 10,2 ; "Steden vernield"
3020  IF s(1)+s(2)=0 THEN PRINT AT 10,2 ; "Afweerraketten
      opgebruikt"
3030  PRINT AT 12,5 ; score ; "neergehaalde raketten"
3040  GO TO 9999
3500  LET xt=m(1,p)+xo
3510  IF xt > 7 AND xt < 29 THEN LET xs=8 : GO SUB kilsilo
3520  IF xt > 223 AND xt < 245 THEN LET xs=224 : GO SUB
      kilsilo
3530  IF xt > 55 AND xt < 93 THEN LET xs=56 : GO SUB kilcity
3540  IF xt > 143 AND xt < 181 THEN LET xs=144 : GO SUB
      kilcity
3550  RETURN
4000  IF xs=8 THEN LET s(1)=0
4010  IF xs=224 THEN LET s(2)=0
4020  PRINT AT 21,xs/8 ; "□□□"
4030  RETURN
4500  LET city=city-1
4510  FOR r=19 TO 21
4520  PRINT AT r,xs/8 ; "□□□□□"
4530  NEXT r
4540  RETURN
5000  IF INKEY$=" " THEN RETURN
5010  LET d$=INKEY$
5015  IF CODE d$ < 49 OR CODE d$ > 51 THEN RETURN
5020  GO SUB fire
5030  RETURN
```

"TOP - DOWN" ANALYSE

De meeste programma's uit dit boek zijn nogal eenvoudig en kunnen gemakkelijk worden gevolgd. Dit programma is ingewikkelder en verdient enkele toelichtingen omdat hier een veel gebruikte en machtige techniek werd aangewend die men wel eens *top - down analyse* of *stapsgewijze verfijning* noemt.
Het principe van deze werkwijze steunt op een ontbinding van het programma in een sequentie van een aantal belangrijke stappen. Dan gaat men na of men deze stappen rechtstreeks kan coderen of dat het misschien gemakkelijker is deze stappen in kleinere stapjes op te splitsen. Men past deze werkwijze toe tot alle stappen voldoende klein zijn zodat ze zonder moeite kunnen worden geprogrammeerd. Elke stap wordt op elk niveau een subroutine.
In dit programma ziet de eerste stap van de top-down analyse er ongeveer zo uit :

Initialiseer variabelen
Teken steden en silo's

Zolang er afweerraketten en steden zijn
- Genereer nieuwe raketten
- Beweeg afgevuurde raketten

Print de score uit

Als we de listing even bekijken, zien we dat de initialiseringen worden uitgevoerd in de regels 1 tot en met 80. Dan verwacht U een instructie als:

```
100  GO SUB 500
```

om de display (uitlezing) routine in te voeren; maar U stelt vast dat dit als volgt wordt geschreven:

```
100  GO SUB display
```

en dat "display" een variabele is die in het initialiseringsdeel op 500 werd geset. Hierdoor wordt het programma beter leesbaar en dit is bijzonder handig bij het eventueel debuggen.
Regels 110 tot en met 160 vormen de "zolang . . ." - lus uit de eerste stap van onze programma-opsplitsing. U vindt in deze lus twee subroutine - aanroepen: één om nieuwe raketten voort te brengen (genmis) en één om de afgevuurde raketten te laten bewegen (movmis).
We kijken even naar "movmis". Opdat we een raket zouden kunnen verplaatsen, moeten we eerst weten waar ze zich bevindt. Er zijn op elk moment slechts 20 raketten die met behulp van 3 informatie-items (hun x-coördinaat, hun y-coördinaat en de aanvalshoek) in een array worden opgeslagen. We spreken af dat indien de aanvalshoek in een vakje nul is, zich op die plaats geen raket bevindt. Op deze manier hoeven we slechts één van de drie parameters uit te testen om te weten te komen of er al dan niet op die plaats een raket is. Bovendien moeten we dan, na het onschadelijk maken van een raket, slechts de aanvalshoek gelijkstellen aan nul om aan te duiden dat die raket er niet meer is.
De opsplitsing van "movmis" ziet er dan zo uit:

Voor alle mogelijk posities van de aanvallende raketten.
- Indien er geen raket is, sla de rest van de lus over
- Bereken nieuwe coördinaten voor de raket.
- Indien de raket nu uit beeld is, test dan of ze een stad/silo heeft geraakt en sla de rest van de lus over.
- Plot de nieuwe raketpositie uit.
- Test of het toetsenbord werd gebruikt.

Als U naar de listing kijkt zult U merken dat twee van deze blokken werden geschreven als subroutines: de "test op het treffen van een stad/silo" (hitest) en de "test op het aanraken van het toetsenbord" (kbhit). Deze laatste routine werd voorzien omdat we moeten weten of de gebruiker het toetsenbord kort geleden heeft aangeraakt om een silo te activeren of niet.
Kijk nu even naar kbhit:

Indien geen toets werd ingedrukt, dan RETURN.
Indien een andere toets dan "1" of "2" werd ingedrukt, dan RETURN.
Vuur een afweerraket af.

en U zult vaststellen dat het "vuren" (fire) weer een andere subroutine is.
Met deze manier van werken hoeven we ons slechts met kleine beetjes tegelijk bezig te houden. We laten de reconstructie van genmis, fire en prnscr aan Uw ontdekkingszin over.

AANPASSINGEN EN VERBETERINGEN

1. Op het ogenblik dat U een raketsilo activeert, zegt het programma niet over welke silo het gaat. Zorg ervoor dat de geactiveerde silo begint te flitsen (en dat hij na het vuren wordt geïnactiveerd).
2. De banen van raketten en afweerraketten (NB. hou er rekening mee dat bij het tekenen van nieuwe banen de kleuren van stukjes van oude banen kunnen veranderen; hier kunt U niets aan doen).
3. Een geraffineerde versie van dit spelletje gebruikt MIRV's als aanvalswapens die bij het neerkomen uiteenvallen in meerdere springkoppen
4. De steden worden op een nogal simplistische manier vernield (ze worden letterlijk weggeveegd). Probeer eens een paddestoelwolk en wat puinhopen al te beelden.
5. Gebruik ook wat geluidseffecten. (Denk eraan dat hierdoor het spel aanzienlijk wordt vertraagd)
6. Soms duurt het lang voordat het programma reageert op het indrukken van toets "1" of "2", de oorzaak is dat kbhit maar weinig frequent wordt aangeroepen. Experimenteer of U "GO SUB kbhit" niet naar andere plaatsen in het programma kunt verhuizen zodat de reactietijd verbetert zonder dat daardoor de rest van de display op dramatische wijze vertraagt.

22. PEEK EN POKE

We moeten een elementaire kennis van het binaire talstelsel hebben om te kunnen begrijpen dat computers met *bits* werken. Eigenlijk werken ze met groepjes van bits die men *bytes* noemt. Eén byte is een sequentie van acht bits: 10110001 en 00111011 zijn bytes.
Er bestaan 256 verschillende bytes; indien U ze omzet in decimale getallen dan zijn dit de getallen van 0 tot en met 255. Misschien herkent U deze getallen: de codes voor de Spectrumkarakters zijn de getallen tussen 0 en 255. Inderdaad, elk karakter wordt voorgesteld door precies één byte.
Een programma (en bepaalde stappen die worden gezet in de uitvoering ervan) is niets anders dan een opeenvolging van karakters; de machine kan dus het programma opslaan in de vorm van een sequentie van bytes. Om de juiste volgorde te verzekeren krijgt iedere byte een referentiegetal dat men het *adres* van die byte noemt.
U moet zich het programma in de machine dus voorstellen als :

Adres	Byte
1	11001100
2	01110000
3	00000000
4	11101111
.	
.	

lammer genoeg maakt de "machine-architectuur" een dergelijk eenvoudig schema onmogelijk. De Spectrum bestaat uit een aantal microchips die luisteren naar de namen:

ROM (Read Only Memory)
RAM (Random Access Memory)
CPU (Central Processing Unit)
SCL (Sinclair Logic Chip)

De ROM herbergt de BASIC-interpreter; de RAM slaat uw programma op en alles wat in de loop van het programma moet worden uitgewerkt; de CPU voert de rekenkunde en de logica uit: de SCL organiseert de samenwerking tussen de andere drie.
De CPU en de SCL interesseren ons niet onmiddellijk, maar *wel* de ROM en de RAM. Door PEEK te gebruiken kunnen we precies te weten komen welke bytes op welke adressen in ROM en RAM opgeslagen zitten.
De instructie :

```
PRINT PEEK 837
```

print de byte uit die opgeslagen zit in het adres 837 (dit is 40, of in binaire notatie. 00101000). In werkelijkheid print deze instructie het corresponderend decimaal getal uit, waardoor het geen kwaad kan een byte te beschouwen als een getal tussen 0 en 255.
Waar moeten we op letten om doeltreffend te "PEEKen"?
De ROM-adressen zijn genummerd van 0 tot en met 16383.
Maar de RAM-adressen zijn interessanter. Deze zijn genummerd van 16384 tot en met 23754 (of meer indien een microdrive werd aangesloten) en worden door de machine zelf gebruikt. Het startadres van uw programma wordt in de systeemvariabele PROG opgeslagen. Deze systeemvariabele zit zelf in de adressen 23635 en 23636.
Dus is het startadres voor uw programma :

```
PEEK 23635 + 256 * PEEK 23636
```

Zonder microdrive zou het resultaat 23755 moeten zijn. Probeer dit eens
Om te weten te komen wat in RAM werd opgeslagen, laat U de Spectrum het werk doen door middel van volgend programmaatje :

```
1000  LET q = PEEK 23635 + 256 * PEEK 23636
1010  LET r = PEEK q
1020  IF r > 23 THEN PRINT q ; "□□" : r ; TAB 12 ; CHR$r
1030  IF r < = 23 THEN PRINT q ; "□□" ; r ; TAB 12 ; "?"
1040  LET q = q + 1
1050  GO TO 1010
```

We hebben opzettelijk grote regelnummers gebruikt omdat het de bedoeling is dat voor lagere regelnummers een testprogramma wordt geschreven waarin U, door GO TO 1000 te gebruiken. kunt vaststellen waar en hoe het is opgeslagen.
Typ dus bovenstaand programma en het volgende testprogramma in :

```
10  REM start
20  PRINT AT 0,10 ; "*"
```

Toets nu GO TO 1000 in en kijk wat er gebeurt. U krijgt een uitprinting zoals

deze :

23755	0	?
23756	10	?
23757	7	?
23758	0	?
23759	234	REM
23760	115	s
23761	116	t
23762	97	a
23763	114	r
23764	116	t
23765	13	?
23766	0	?
23767	20	?
23768	23	?
23769	0	?
23770	245	PRINT
23771	172	AT
23772	48	0
23773	14	?
23774	0	?
23775	0	?
23776	0	?

We stellen vast dat ons programma – of tenminste het grootste deel ervan – in de derde kolom te voorschijn komt: REM, s, t, a, r, t, ..., PRINT, AT, 0, ... Maar er is nog meer: de derde kolom lijkt ons niet veel wijzer te maken, maar de tweede doet dit wel. Bijvoorbeeld, in adres 23756 vinden we 10 en in 23767 vinden we 20: waarschijnlijk zijn dit de regelnummers. (Met de lagere regelnummers is het wel een beetje specialer dan het lijkt, en de 7 in 23757 en de 23 in 23768 stellen eigenlijk het aantal karakters voor die voorkomen in de betreffende regels: zie *handboek*.)
De vraagtekens in kolom drie staan er met reden: indien U in regel 1020 "IF r > 23" weglaat en 1030 schrapt, zal het programma niet werken. De reden hiervoor is dat deze karakters *control*-karakters zijn (die bijvoorbeeld kleuren geven) die de machine tracht uit te voeren als commando's. Op deze manier krijgt de machine dikwijls nonsens aangeboden waarop ze met een foutmelding reageert. Maar U kunt met behulp van de karaktertabel in het *handboek* te weten komen welke deze controlkarakters zijn. In adres 23765 hebben we bijvoorbeeld karakternummer 13 en dat is ENTER – inderdaad we *hebben* toen deze toets ingedrukt, nietwaar?
U ziet dat er dus enkele details moeten worden uitgeknobbeld; maar wat belangrijk is, is dat we kunnen zien waar het programma opgeslagen zit. Om nog meer te kunnen zien, drukt U op "y" voor een *scroll* (een eigenschap die ons PEEK programma uitermate nuttig maakt). We krijgen nu :

23777	0	?
23778	0	?

23779	44	,
23780	49	1
23781	48	0
23782	14	?
23783	0	?
23784	0	?
23785	10	?
23786	0	?
23787	0	?
23788	59	;
23789	34	"
23790	42	*
23791	34	"
23795	13	?

en U kunt doorgaan met listen door telkens opnieuw te scrollen, en dit kan een poosje duren. Weldra zal de routine op regel 1000 zichzelf beginnen uitlisten . . .
Probeer zelf enkele programma's op regels 10, 20, 30, enzovoort. Wijzig de regels 1000 tot en met 1040 niet en PEEK uw programma's met GO SUB 1000.
Wat zeer eigenaardig is, is de manier waarop *getallen* worden opgeslagen. De 10 in PRINT AT 10, bijvoorbeeld, blijkt de adressen 23780 tot en met 23787 in beslag te nemen en dit komt ons voor als een nogal ruime spatie voor zo'n klein getal. Inderdaad, maar dit houdt verband met het feit dat de Spectrum kan werken met *drijvende komma* rekenkunde (decimale getallen zoals 23,567) en dat de Spectrum daartoe een geschikte code gebruikt. U kunt er uren plezier aan beleven door uit te zoeken hoe een gegeven getal in werkelijkheid wordt opgeslagen.
Maar het mooist van al is dat U nu weet *waar* een bepaalde programmabyte zich bevindt en U hem dus kunt *vervangen* door een andere. De instructie die U toelaat dit te doen is POKE.
We kijken even welk effect POKE teweegbrengt.
Wat doet ons klein programmaatje van hierboven? Het print een * uit op positie 0,10. Run en kijk.
We "POKEn" dit programmaatje door de volgende regels toe te voegen:

```
30  POKE 23790,96
40  GO TO 10
```

Run dit nu door het volgende in te typen :

```
GO TO 30
```

In plaats van een * wordt een £ - symbool uitgeprint.
Waarom? Wel, omdat regel 30 in adres 23790 (waar oorspronkelijk * zat opgeslagen) het nieuwe karakter met code 96 heeft "gePOKEt", en dit is £.
Als U een listing maakt zult U vaststellen dat regel 20 van het oorspronkelijk programma veranderd is in :

```
20  PRINT AT 0,10 ; "£"
```

En we experimenteren maar door! BREAK en laat regels 30 en 40 verdwijnen. Breng met behulp van EDIT regel 20 terug in zijn oorspronkelijke staat, maar verander de 0 in een 1, dus :

```
20  PRINT AT 1,10 : "*"
```

Nu doet U: GO TO 1000 en PEEKt alles opnieuw. Wanneer U van het scherm afraakt, scrollt U zodat U verder kunt gaan.
U zult de verandering tegenkomen. De adressen 23772 en 23776 zijn nu :

23772	49	1
23776	1	?

geworden, voor de rest is alles bij het oude gebleven.
Om uw kennis te toetsen, voegt U volgende programmaregels toe :

```
30  POKE 23772,49
40  POKE 23776,1
50  GO TO 20
```

Maak van regel 20 opnieuw: 20 PRINT AT 0,10 ; "*". Druk nu op RUN.
Nu ziet U op het scherm *twee* sterretjes (namelijk één op lijn 0 en één op lijn 1 van het scherm) omdat de machine het eerste programma doorloopt waarbij op lijn 0 een sterretje wordt uitgeprint, daarna POKEt ze de waarden die nodig zijn voor de uitprinting op lijn 1 en tenslotte keert ze naar regel 20 (GO TO 20) terug en wordt op de nieuwe lijn een tweede sterretje gePRINT.
Probeer eens met LIST; eens te meer zult U merken dat regel 20 veranderd werd in: 20 PRINT AT 1,10 ; "*"!
Natuurlijk kunt U POKE niet toepassen op ROM. ROM staat immers voor "read only memory" en dit betekent "*lees*geheugen" en niet dit geheugen kunt U niets anders doen dan *lezen*! Dit is maar goed ook, want anders zou een ongelukkige POKE - instructie de BASIC - interpreter kunnen vernietigen.
De POKE - instructie opent wijde perspectieven, maar om ze juist te gebruiken moet U vertrouwd geraken met de details van de interne codes die de Spectrum gebruikt. De hoofdstukken 24 en 25 van het *handboek* geven U de nodige informatie hierover.

23. TIPS

RAADSELACHTIGE STRUIKELBLOKKEN

1. Om het kwadraat van een getal x te berekenen werkt men meestal x ↑ 2 uit. Voor een positieve waarde van x is dit O K ; maar hoewel ook negatieve getallen kunnen worden gekwadrateerd, zal de notatie met het opstaande pijltje in dit geval geen resultaat opleveren. (Omdat x ↑ a wordt uitgewerkt als EXP(a * LNx) en de logaritme van een negatief getal niet is gedefinieerd.) Indien x negatief is, gebruikt men x * x om het kwadraat te berekenen
2. De berekening van x ↑ 2 indien x een geheel getal is, is eveneens – om analoge redenen – een slechte gewoonte omdat het resultaat niet exact is. Op deze manier kunt U met dezelfde moeilijkheden te maken krijgen die in Debugging IV werden besproken. Hetzelfde geldt trouwens voor x ↑ 3 : het *is* soms beter om te schrijven x * x * x
3. Indien U in een multistatement regel een REM gebruikt, denk er dan aan dat *alles* wat na dit woordje (in die regel) staat door de computer wordt genegeerd. Dus :

```
10  REM start : LET x = 99
```

heeft geen enkel effect, maar:

```
10  LET x=99 . REM start
```

kent de waarde 99 toe aan x.

4. Alle IF/THEN - commando's werken als volgt: indien aan de IF - voorwaarde is voldaan, wordt de *volledige* regel na THEN uitgevoerd, maar indien de IF - voorwaarde niet opgaat, wordt *geen* van de statements na THEN uitgevoerd. Dus:

```
10  IF x=0 THEN LET y=0 : IF x < > 0 THEN LET y=1
```

zal y gelijkstellen aan 0 indien x=0, maar er zal *helemaal niets* gebeuren indien x < > 0. Met aparte regels daarentegen:

```
10  IF x=0 THEN LET y=0
20  IF x < > 0 THEN LET y=1
```

zal y de waarde 1 krijgen indien x verschillend is van 0. Indien uw voorwaardelijke statements in de war raken, kunt U het beste eerst de voorwaardelijke multistatement regels nakijken!

5. De machine werkt in een IF/THEN - statement de *gehele* uitdrukking tussen IF en THEN uit, daarna zal ze pas nakijken of aan deze voorwaarde wordt voldaan. Hierdoor kan het programma fout lopen en kunnen onverwachte foutrapporteringen worden uitgebracht. Bijvoorbeeld:

```
10  IF x$ < > " " AND VAL x$ = 5 THEN STOP
```

veroorzaakt een fout indien x$ leeg is – zelfs indien het x$ < > " " - deel vals is zal de volledige voorwaarde vals zijn, wat VAL x$ ook moge zijn. De machine *blijft* ondanks alles VAL x$ berekenen indien x$ = " ". Het gevaar bestaat dat U *denkt* dat U het programma tegen ongelukjes heeft beveiligd, maar U ziet dat vergissingen mogelijk zijn.

6. Men kan op twee manieren een blanco spatie verkrijgen: enerzijds met de SPACE - toets en anderzijds met de grafiek 8 én CAPS SHIFT. *Deze twee karakters zijn verschillend* voor de Spectrum. Indien U dus spaties opspoort met het commando:

```
10  IF n$(i)="□" THEN GO TO 5000
```

zal dit geen resultaat geven indien de spatie in n$(i) niet van dezelfde soort is als die in regel 10 van uw programma werd gebruikt. Jammer genoeg kunt U de fout niet vinden met behulp van een listing!

7. U kunt een tekening "SAVEn" door middel van:

```
SAVE "Tekening" SCREEN$
```

en als U dat wilt, eenvoudig laden door:

```
LOAD "Tekening"
```

in dit geval zult U de melding "Bytes: Tekening" zien verschijnen, maar niets anders. U moet natuurlijk het volgende doen:

```
LOAD "Tekening" SCREEN$
```

8. Wanneer U bij het laden van een programma in de computer (d.m.v. LOAD) een speciale hoge "bleeptoon" hoort wanneer de machine een programmanaam tegenkomt en indien U bij het horen van deze "bleep" niets op het scherm ziet verschijnen dan weet U dat er iets verkeerd is gelopen.

9. Indien U een programma in het geheugen heeft zitten dat U op tape wilt bewa-

ren (d.m.v. SAVE) en U weet niet meer waar U zich op de tape bevindt, dan loopt U het risico dat U een te groot stuk van de band onbenut laat of, erger, dat U een belangrijk stuk overschrijft. Typ daarom :

VERIFY "Rotzooi"

waarbij "Rotzooi" een naam is die U niet heeft gebruikt voor één of ander programma. Dan laat U de tape lopen en kijkt U de meldingen na zodat U ziet waar op de band plaats is voor uw programma. Wanneer U de geschikte plaats hebt gevonden stopt U de tape, BREAK en LOAD uw programma op de gewone manier.

TRUUKJES MET SYSTEEMVARIABELEN

1. U kunt een veel betere bleep produceren door een rechtstreekse invoer via het toetsenbord te doen, met bijvoorbeeld :

   POKE 23609,50

   Andere getallen dan 50 geven lichtjes verschillende effecten.

2. U kunt de "auto - repeat" versnellen door bijvoorbeeld :

   POKE 23562,2

   rechtstreeks in te voeren. Door 2 te vervangen door 1 of 3 versnelt of vertraagt U de "auto-repeat"; indien U de 2 vervangt door 0 wordt de "auto-repeat" uitgeschakeld.

3. U kunt de wachttijd van de machine voor een "auto-repeat" inkorten door bijvoorbeeld :

   POKE 23561,20

   in te toetsen. U verkort de wachttijd door de 20 te verkleinen en U vergroot hem indien U het getal groter maakt.

4. Bepaalde programma's doen een beroep op een automatische SCROLL - operatie, maar er bestaat geen SCROLL - commando op de Spectrum.
   U kunt dit SCROLL - commando vervangen door een routine :

   ```
   1000  PRINT AT 21,0
   1010  POKE 23692,2
   1020  PRINT
   ```

   De POKE 23692 - instructie met een getal groter dan 1 op het ogenblik dat het scherm volraakt, veroorzaakt een scroll zonder ingang via het toetsenbord, wanneer de uitprinting van een nieuwe regel volgt.

5. Het DRAW x, y - commando – hoe mooi het ook is – tekent een kromme vanuit de actuele positie x0,y0 naar de nieuwe positie x0 + x,y0 + y. Dit betekent dat x en y de incrementen zijn van de oude coördinaten, maar zeker niet de nieuwe coördinaten. U kunt echter vanuit een actuele positie een kromme tekenen naar een nieuwe positie x,y met het commando :

   DRAW x – PEEK 23677, y – PEEK 23678

   Deze methode biedt het voordeel dat ze nog werkt zelfs indien U niet meer weet waar de laatste PLOT - positie was en ze is niet onderhevig aan cumulatieve afrondingsfouten die zich manifesteren indien U een lus gebruikt om krommen te tekenen.
   De adressen 23677 en 23678 bevatten de coördinaten van het laatst uitgeplotte punt!

24. KANT - EN - KLAAR - PROGRAMMA'S

Het ligt in onze bedoeling om met de programma's die nu volgen de verschillende eigenschappen van de Spectrum te illustreren en aan te tonen wat de Spectrum zo al kan realiseren. Elk programma is volledig en moet slechts worden gecopieerd en gerund. Het is niet absoluut nodig de commando's uit de listings te begrijpen, maar indien U dit boek hebt doorgewerkt, moet U in staat zijn de manier waarop de programma's functioneren te analyseren. We hebben in deze listings de grafische karakters getekend zoals ze op het toetsenbord voorkomen. Een vierkantje rond een letter, zoals [H], betekent inverse video; een open vierkantje □ duidt een spatie aan. Vanzelfsprekende spaties werden niet expliciet aangeduid, maar belangrijke spaties die gemakkelijk kunnen worden vergeten *wel*.

DE OPPERVLAKTE VAN EEN DRIEHOEK

Indien a, b en c de zijden zijn van een driehoek, dan wordt de oppervlakte ervan gegeven door de formule $\sqrt{s(s-a)(s-b)(s-c)}$ waarin $s = 1/2(a+b+c)$. Dit programma maakt gebruik van deze formule om de oppervlakte van een driehoek te berekenen waarvan de zijden achtereenvolgens worden ingevoerd.

```
10  INPUT a
20  PRINT "a=□" ; a
30  INPUT b
40  PRINT "b=□" ; b
50  INPUT c
60  PRINT "c=□" ; c
70  LET s=.5*(a+b+c)
75  LET x=s*(s-a)*(s-b)*(s-c)
80  IF x < 0 THEN GO TO 110
90  PRINT "OPPERVLAKTE IS □" ; SQR x
100 STOP
110 PRINT "DEZE ZIJDEN VORMEN GEEN DRIEHOEK"
```

SALDO VAN UW BANKREKENING

```
10  PRINT "VORIG SALDO BEDRAAGT □" ;
20  INPUT s
30  PRINT s
40  PRINT "DEBETLIST"
50  INPUT d
60  IF d < 0 THEN GO TO 90
70  LET s=s-d
80  GO TO 50
90  PRINT "CREDITLIST"
100 INPUT d
110 IF d < 0 THEN GO TO 140
```

```
120  LET s=s+d
130  GO TO 100
140  PRINT "HUIDIG SALDO IS □", s
```

Opmerking : Indien U de volgende twee regeltjes toevoegt :

```
 55  PRINT d
105  PRINT d
```

dan zal de machine deze items listen.

TIJGERJACHT

```
     PAPER 7 , INK 8 : BORDER 1
 10  INPUT "Kies de grootte van het bos  max 16 □" ; w
 15  IF w > 16 THEN LET w=16
 20  FOR i=0 TO 8 * w STEP 8
 30  PLOT i+64,32
 40  DRAW 0,8 * w
 50  PLOT 64,i+32
 60  DRAW 8 * w,0
 70  NEXT i
 80  LET z$="0123456789111111"
 90  LET y$="□□□□□□□□□□0123456"
100  PRINT AT 19,8 ; z$ (TO w)+"□" +"x"
110  PRINT AT 20,8 ; y$ (TO w)
120  PRINT AT 16-w,6 , "y" ; AT 17-w,0
130  FOR i=1 TO w
140  PRINT TAB 6-(w-i > 9) : w-i
150  NEXT i
200  LET mx=INT(w * RND)
210  LET my=INT(w * RND)
220  INPUT "Waar zit de tijger? □" ; x ; "□" ; y
225  IF x > w OR y > w THEN GO TO 220
230  LET d=((mx-x) * (mx-x)+(my-y) * (my-y)) / w / w
240  LET d=d * 8  : IF d > 4 THEN LET d=4
250  PRINT PAPER d+2 ; AT 17-y, 8+x ; OVER 1 , "□"
255  IF mx=x AND my=y THEN GO TO 300
260  GO TO 220
300  PRINT AT 17-my,8+mx : OVER 1 ; "*"
310  PRINT AT 0,0 ; FLASH 1 ; "Hebbes!"
320  FOR i=1 TO 30
330  BEEP .1 / i,i
340  NEXT i
```

Opmerkingen :

1. U kunt de grootte van het woud kiezen tussen 1 en 16. Indien U een grotere waarde kiest, neemt de computer 16. Het woud wordt getekend met behulp van x- en y-coördinaten langs de onderkant en aan de zijkant (regels 20 - 70 worden gebruikt voor het woud en 80 - 150 voor de coördinaten). Merk op dat de tekening maken het grootste deel van het programma in beslag neemt!
2. Wanneer wordt gevraagd "Waar zit de tijger?" dan moet U twee getallen tussen 0 en w – 1 invoeren; dit zijn de coördinaten van uw gissing. Indien U getallen invoert die niet in dit bereik liggen, dan worden ze genegeerd.
3. Vervolgens print de machine op deze positie een gekleurd vierkantje. De kleur ervan geeft U een aanwijzing over hoe dicht U zich in de buurt van de tijger bevindt: rood betekent erg dicht, magenta vrij dicht, groen vrij ver af en geel erg ver af.
4. U gaat door tot U de tijger hebt kunnen localiseren. Dan krijgt U een "*", een flitsende gelukwens, en een stukje muziek.

COMPONIST

Dit programma laat de Spectrum muziek spelen. De noten worden één voor één ingevoerd als enkelvoudige letters of als enkelvoudige letters, gevolgd door een #- teken om de halve tonen aan te geven. Voer * * in om de sequentie van noten te beëindigen en de Spectrum het deuntje te laten spelen.

```
10   DIM s(7)
11   DIM n$(2)
12   DIM t(1500)
20   LET s(1)=-3 : LET s(2)=-1 : LET s(3)=0 : LET s(4)=2
30   LET s(5)=4 : LET s(6)=5 : LET s(7)=7
35   FOR q=1 TO 1500
40   INPUT "Voer een noot in" ; n$
42   IF n$="* * " THEN GO TO 200
45   LET i=0
50   LET p=CODE n$-64
55   IF p > 10 THEN LET p=p-32
60   IF n$(2)="#" THEN LET i=1
100  LET t(q)=s(p)+i
110  NEXT q
200  FOR r=1 TO q
210  BEEP 0.5, t(r)
220  NEXT r
```

De details over de manier van werken vindt U in de paragraaf over arrays.

Verbeteringen en aanpassingen

1. Vervang regel 40 door :

```
40   LET n$=CHR$(INT(RND * 7)+65)
```

Het programma componeert nu zelf een "muziekstuk" dat maar liefst 1500 no-

ten bevat. Er komen geen halve tonen in voor, zodat het resultaat nogal bot klinkt.

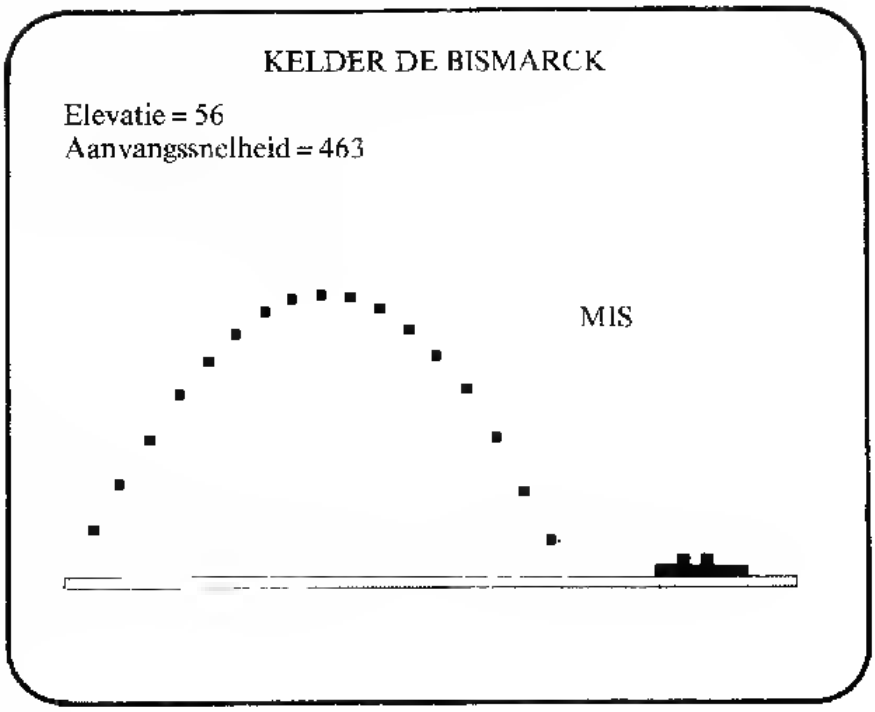
2. De tijd van elke noot werd gefixeerd op een halve seconde. Kunt U een routine maken die de gebruiker in staat stelt – tegelijk met de noot – de tijd van elke noot in te voeren? U zult hiervoor nog een extra array nodig hebben waarin de waarden van de tijd kunnen worden opgeslagen en deze array moet dezelfde lengte hebben als t. (Met 16 K zullen twee arrays met lengte 1500 niet in het geheugen kunnen. Maak daarom twee arrays niet lengte 750.)
3. Ontwerp een gelijksoortige techniek zodat de toevallige muziek die in (1) wordt voortgebracht ook noten gebruikt met toevallige tijden.
4. De muziek noot na noot invoeren kan soms een nogal omslachtige bezigheid zijn. Kunt U een methode ontwerpen waarmee ze als één enkele string wordt ingevoerd?
5. Dit programma zou een nuttig componeerinstrument zijn indien de noten op een gemakkelijke manier konden worden aangepast. Probeer eens of U een "melodie editor" (meloditor) kunt uitvinden die het mogelijk maakt dat – door bijvoorbeeld 38,A# in te voeren – de achtendertigste noot wordt veranderd in een A#.
6. Verruim de één-octaaf-beperking. U zult verschillende ingangen per noot nodig hebben. A,0 kan bijvoorbeeld betekenen "A in het octaaf van de grond-C" en C#,2 kan betekenen "C#, twee octaven hoger dan de grond C".

KELDER DE BISMARCK

Een patrouillevaartuig voert een routine-opdracht uit in de Noordzee. Plotseling doemt een vijandelijk schip op uit de mist op nog geen mijl afstand. U geeft de kanonniers de opdracht de elevatie en de aanvangssnelheid van hun geschut in te stellen. Zult U de *Bismarck* kunnen kelderen?

```
10  PRINT TAB 8 ; "KELDER DE BISMARCK"
20  LET t = 15 * (1 + RND)
30  PRINT AT 21,0 : INK 5 ; INVERSE 1 ; "□□□□□□□□□□□□
    □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□"
40  PRINT AT 20,t – 2 ; INK 4 ; "▟▙▟▙"
50  INPUT "Elevatie = □" ; e
60  PRINT AT 3,0 : "Elevatie = □" ; e
70  INPUT "Aanvangssnelheid = □" : v
80  PRINT "Aanvangssnelheid = □" ; v
90  LET a = v * COS (PI * e/180)
100 LET b = v * SIN (PI * e/180)
110 FOR j = 0 TO b/16 STEP .3
115 LET c = .01 * (b * j – 16 * j * j)
120 IF a * j > 6200 THEN GO TO 190
130 IF c > 40 THEN GO TO 170
140 INK 2
150 PLOT .04 * a * j,4 * c + 8
160 BEEP .005,c + 10
```

```
170  NEXT j
180  IF ABS (a * b/3200 - t) < 3 THEN GO TO 210
190  PRINT AT 10,20 ; "MIS!"
200  STOP
210  FOR j = 0 TO 15
220  PRINT AT 20 - j,t - 2 ; "blup!" : BEEP .3,6 - 3 * j
230  NEXT j
```



Opmerkingen :

1. De Spectrum begint met de zee en een toevallig geplaatst silhouet van een schip te tekenen. Regel 20 kiest de toevallige positie, 30 print de zee en 40 het schip.
2. Regels 50 - 80 voeren de elevatie (e) en de aanvangssnelheid (v) in en schrijven ze op het scherm. Dan vraagt de machine de speler deze waarden via het toetsenbord in te voeren. e is in graden en deze hoek moet groter zijn dan 0° en kleiner dan 90° ; v is in voet per seconde en moet positief zijn.
3. Regels 70 - 170 herekenen en tekenen de baan van de granaat met een interval van 0.3 seconden (zie onder wiskundige opmerkingen)
4. Regel 180 vertelt U dat het schip werd geraakt indien uw granaat de waterlijn treft binnen een afstand van 600 voet midscheeps. Om het spelletje moeilijker te maken kunt U de 3 in 2 of in 1.5 veranderen, of om het gemakkelijker te maken in 4 of in 5.
5. Regels 190 - 230 zijn uitvoerroutines voor het geval van een voltreffer of een misser.
6. Om te beginnen of om opnieuw te beginnen drukt U zoals gewoonlijk op

RUN en daarna op ENTER.

7. Het scherm is, zoals het hier werd gesimuleerd. 6400 voet breed.
8. Het programma is nogal "gebruiker - vriendelijk" en blijft zelfs werken indien de baan van het projectiel de bovenkant van het scherm overschrijdt (dit dankzij regel 120).
9. De grafiek uit regel 40 wordt gemaakt met de toetsen 3 en 1 in de CAPS SHIFT - positie.
10. "PI" in regels 90 en 100 vormt U met toets M in de extended mode en *niet* met de toetsen P en I!

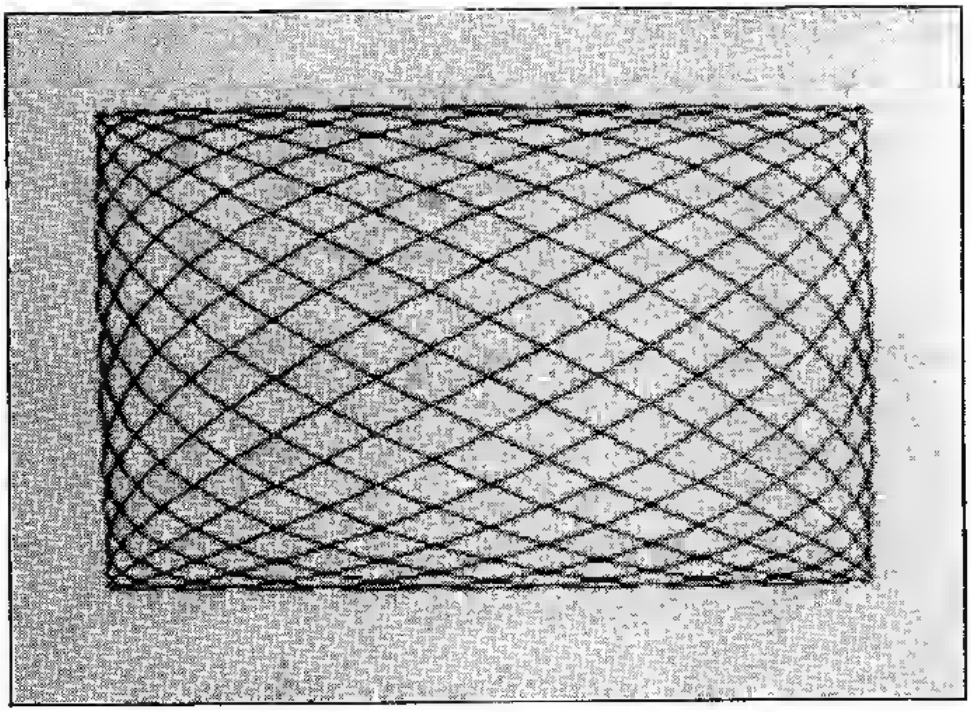
Wiskundige opmerkingen

De baan van het projectiel wordt berekend aan de hand van een wiskundige formule (de luchtweerstand wordt hierin evenwel verwaarloosd) die het traject geeft van een bewegend lichaam onder invloed van de gravitatie (die wordt verondersteld 32 voet per seconde kwadraat te zijn). In regel 110 stelt j de tijd voor en is a de horizontale en b de vertikale component van de snelheid. De uitdrukking PI * e/180 zet het aantal graden om in radialen. Regel 150 berekent de hoogte op het ogenblik j en zet ze om in coördinaten van het scherm (elke PLOT - pixel heeft, omgerekend een oppervlakte van 25 vierkante voet)
In regel 110 is b/16 de tijd die nodig is om de waterlijn te raken.
In regel 180 is a * b/1600 de afstand die werd afgelegd; hierin wordt 3200 gebruikt omdat we een PRINT - statement gebruiken in regel 40, dus bedraagt de afstand tot het schip 200 * t.

FIGUREN VAN LISSAJOUS

De figuren van Lissajous zijn een andere interessante illustratie van het uitplotten van curves. Om ze te tekenen maakt men gebruik van, wat de wiskundige, een *parametervergelijking* noemt. Dit betekent dat U x en y afhankelijk maakt van een variabele t en dat U x en y uitplot in functie van een sequentie van waarden voor t.

```
10  INPUT "Eerste getal" : p
20  INPUT "Tweede getal" ; q
30  INPUT "Faseverschuiving" ; r
40  LET t=0
50  LET m=p
60  IF q < p THEN LET m=q
70  LET x=127+120 * COS (t * PI/180 * p+r)
80  LET y=87+80 * SIN (t * PI/180 * q)
90  IF t=0 THEN PLOT x,y
100 IF t > 0 THEN DRAW x – PEEK 23677,y – PEEK 23678
110 LET t=t+10/m
120 GO TO 70
```



Opmerkingen

1. Om mooie resultaten te krijgen moeten p en q gehele getallen zijn.
2. r mag willekeurig worden gekozen, maar tussen 0 en 3 is het beste.
3. Een goed begin hebt U met p = 5, q = 7 en r = 1.
4. PI is de toets "M" in extended mode.
5. Om op te houden drukt U op BREAK met CAPS SHIFT.
6. Meer spectaculaire resultaten (maar langere uitplottijden) krijgt U met p = 31, q = 29 en r = 0.
7. Regel 100 verzorgt de overgang van het oude uitplotpunt naar het nieuwe en wel zo dat DRAW een lijn kan trekken tussen beide. De systeemvariabelen in de adressen 23677 en 23678 die worden "gePEEKt", bevatten de laatste PLOT - coördinaten.

DOBBELSTENEN GOOIEN

Bij het gooien van twee dobbelstenen zult U al wel hebben opgemerkt dat het totaal 7, meer voorkomt dan enig ander totaal. Theoretisch zou, na 36 worpen, het totaal

2 gemiddeld 1 maal
3 gemiddeld 2 maal
4 gemiddeld 3 maal
5 gemiddeld 4 maal
6 gemiddeld 5 maal
7 gemiddeld 6 maal
8 gemiddeld 5 maal
9 gemiddeld 4 maal
10 gemiddeld 3 maal
11 gemiddeld 2 maal
12 gemiddeld 1 maal

moeten voorkomen. In de praktijk zal dit natuurlijk niet na 36 worpen voorkomen, maar op de lange duur zult U zien dat dit gemiddelde zeer dicht wordt benaderd.
U kunt met de Spectrum de wereld van de statistiek leren ontdekken en de machine een *simulatie* van het dobbelsteenspel laten uitvoeren door gebruik te maken van toevalsgetallen. Dit programma "gooit" 144 maal (4 keer 36) met twee dobbelstenen, telt hoe dikwijls het totaal aantal ogen 2, 3, 4, ... enzovoort bedraagt en plot de resultaten uit in een "staafdiagram". Bovendien vergelijkt dit programma het experimenteel resultaat met de te verwachten theoretische frequenties.

```
10   DIM a(11)
20   FOR j = 1 TO 144
30   LET d = 1 + INT(6 * RND) + INT(6 * RND)
40   LET a(d) = a(d) + 1
50   NEXT j
80   FOR j = 2 TO 12
90   LET q = a(j - 1)
100  LET q0 = INT(q/2)
110  LET q1 = q - 2 * q0
120  FOR t = 1 TO q0
130  PRINT AT 18 - t,4 + 2 * j ; "■"
140  NEXT t
150  IF q1 = 1 THEN PRINT AT 17 - q0,4 + 2 * j ; "▗"
160  NEXT j
170  PRINT AT 19,8 ; "2 □ 3 □ 4 □ 5 □ 6 □ 7 □ 8 □ 9 □ 1 □ 1 □ 1 "
180  PRINT AT 20,24 ; "0 □ 1 □ 2"
```

Opmerking

Het staafdiagram stelt de relatieve frequentie voor van de totalen 2, 3, 4, ..., 12. Het theoretisch diagram vertoont een driehoekige vorm met een top bij 7. Hoe dicht benadert het experiment het theoretisch resultaat? Krijgt U na een tweede (eventueel derde, enzovoort) RUN hetzelfde resultaat? Waarom niet?

TOETS VAN DE REKENVAARDIGHEID

Dit programma geeft de operator (uw zeven jaar oude spruit) een optellingetje van twee toevallige getallen van twee cijfers, kijkt na of het antwoord juist of verkeerd is en legt uit, indien het antwoord verkeerd is, hoe de som moest worden gemaakt.

```
10   LET v = 10
20   LET c = 0
30   PRINT "Ik ben Sinclair Spectrum. Wie ben jij?"
40   INPUT e$
50   PRINT "OK, □" ; e$ ; "□ kun jij het volgende"
60   PRINT "eens voor mij oplossen?"
70   LET x = INT (v * v * RND)
80   LET y = INT (v * v * RND)
90   IF x + y <= 12 THEN GO TO 7 * v
```

```
100  LET x1 = INT(x/v)
110  LET x2 = x - v * x1
120  LET y1 = INT(y/v)
130  LET y2 = y - v * y1
140  IF x2 + y2 > = v THEN LET c = 1
150  PRINT x ; "+" ; y ; "□=□?□" ;
160  INPUT a
170  PRINT a ; "□?"
180  LET b = x + y - a
190  PRINT ("juist" AND b = 0) + ("sorry, verkeerd"
     AND b < > 0)
200  IF b = 0 THEN GO TO 190
210  PRINT x2 ; "+" ; y2 ; "□ is □" ; x2 + y2 - c * v :
220  IF c = 1 THEN PRINT "□ overdracht 1"
230  PRINT "□ en dan □" ; x1 ; "+" ; y1 ; "+ de overdracht"
     AND c = 1 ; "□ is □" ; x1 + y1 + c
240  PRINT "dit geeft □" ; x + y
```

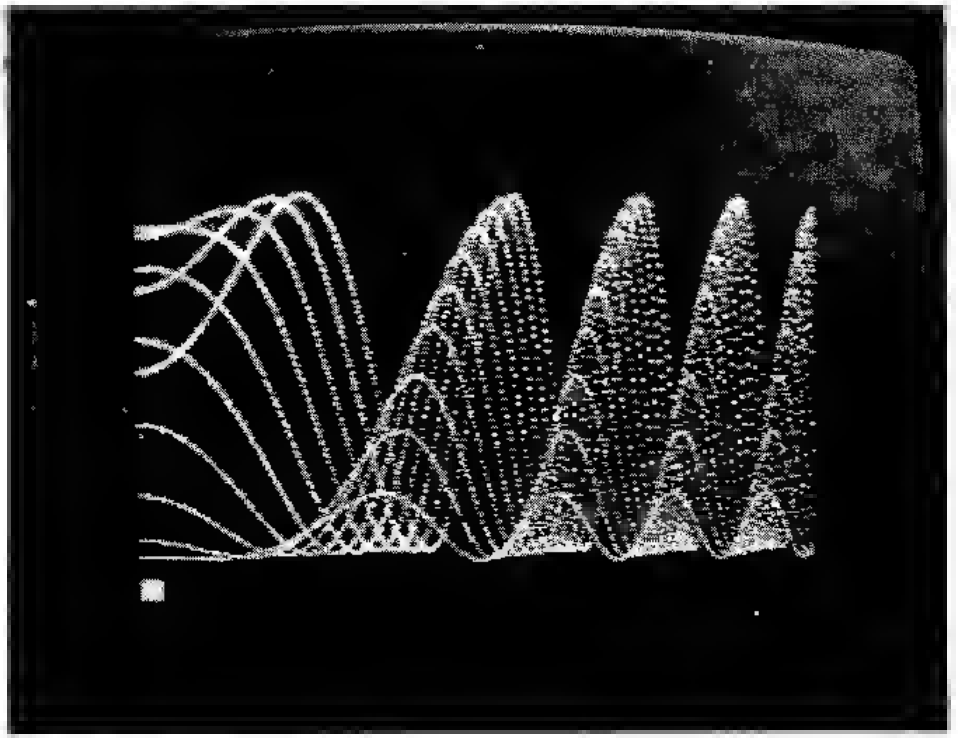
Opmerkingen :

1. Regel 30 is een "vertrouwen - opwekker". Hij informeert naar de naam van de operator en regel 50 gebruikt deze naam in een vraag.
2. In regel 160 vertelt de operator de computer wat hij denkt dat het antwoord is.
3. Vanaf regel 200 legt de computer de operator die een verkeerd antwoord gaf uit hoe de som moest worden gemaakt.

GRAFIEKDEMONSTRATIE 1

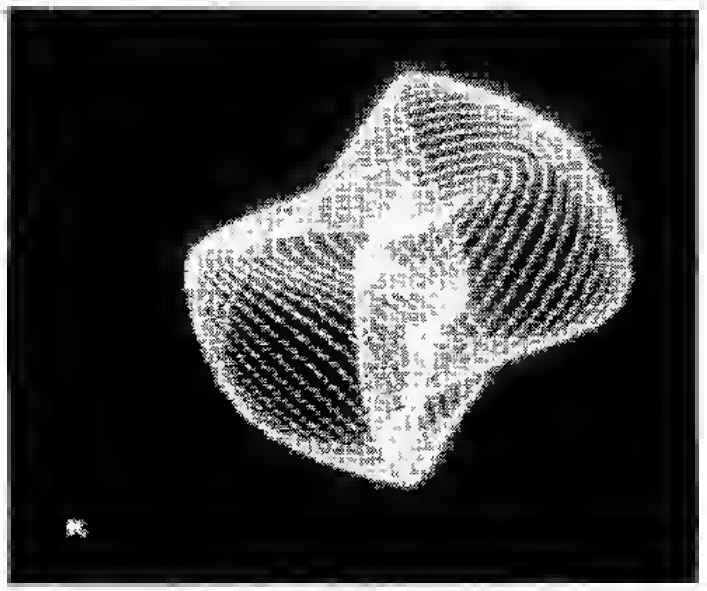
Dit erg korte programma geeft een heel mooi beeld op het scherm

```
     BORDER 0 : PAPER 0
10   FOR j = 0 TO 9
20   INK 7 - INT (j/2)
30   FOR i = 0 TO 510
40   PLOT .5 * i,(80 + 70 * SIN(i * i/10000) + j/2) * (1 - j * j/100)
50   NEXT i
60   NEXT j
```



GRAFIEKDEMONSTRATIE 2

```
    BORDER 0 : PAPER 0 : INK 7
 5  LET i=0
10  PLOT 128+(30+50 * COS(i/41)) * COS(i/40),
    88+(50+30 * SIN(i/39)) * SIN(i/40)
15  LET i=i+1
20  GO TO 10
```



Opmerkingen :

1. Indien U deze figuur mooi vindt, probeer dan eens een analoge figuur door getallen in de formule te veranderen: in het bijzonder kunt U de getallen 41 en 39 door andere vervangen zonder gevaar te lopen van het scherm af te raken.
2. Indien U niet van zwart en wit houdt, dan kunt U één of andere INK - instructie geven en dan CONTINUE indrukken, de pixels worden dan in de nieuwe INK - kleur uitgeplot.

PRIEMFACTOREN

```
10  INPUT k
30  PRINT k ; "□=□" ;
40  LET k0 = k
50  IF INT(k/2) * 2 = k THEN GO TO 140
60  LET n = 3
70  IF INT(k/n) * n = k THEN GO TO 110
80  IF n * n > k THEN GO TO 170
90  LET n = n + 2
100 GO TO 70
110 PRINT n ; " . " ;
120 LET k = k/n
130 GO TO 70
140 PRINT "2. " ;
150 LET k = k/2
160 GO TO 50
170 IF k = k0 THEN PRINT FLASH 1 ; "PRIEM"
180 IF k < k0 AND k > 1 THEN PRINT k
```

Opmerkingen :

1. Aan INT(k/n) * n = k is slechts voldaan indien k deelbaar is door n; dit is dus een test voor de delers.
2. Het programma poogt k te delen door 2 en door alle oneven getallen die kleiner zijn dan de vierkantswortel van k. Indien k door geen van deze getallen deelbaar is, dan is k een priemgetal.
3. Indien een deler wordt gevonden, deelt het programma k door dit getal en gaat het op zoek naar een nieuwe deler van dezelfde grootte. Indien zo geen deler wordt gevonden, wordt naar een grotere deler gezocht.
4. De machine heeft ongeveer 35 seconden nodig om een priemgetal van 8 cijfers te ondekken, met minder cijfers zal dit zo lang niet duren. (De 35 seconden geldt voor het getal 11111117 , voor andere getallen zal dit natuurlijk lichtjes verschillen.)

DERDE GRAADS VERGELIJKINGEN OPLOSSEN

Dit programma zoekt alle reële wortels van derdegraads vergelijkingen $ax^3 + bx^2 + cx + d = 0$ met een aanvaardbare nauwkeurigheid.

```
10   INPUT a, b, c, d
50   LET b = b/a
60   LET c = c/a
70   LET d = d/a
80   LET x = 0
90   LET g = 2 * x * x * x + b * x * x - d
100  LET h = 3 * x * x + 2 * b * x + c
110  IF h = 0 THEN GO TO 200
120  IF ABS(x - (g/h)) < 1.E - 8 THEN GO TO 300
130  LET x = g/h
140  GO TO 90
200  LET x = x + 1
210  GO TO 90
300  PRINT "x1 = □" ; x
400  LET a = b + x
410  LET b = x * x + b * x + c
420  LET d = a * a - 4 * b
430  IF d < 0 THEN PRINT "Andere wortels zijn imaginair"
440  IF ABSd < 1.E - 7 THEN PRINT "Mogelijke numerieke instabiliteit"
445  IF d < 0 THEN STOP
450  PRINT "x2 = □" : (-a + SQRd) / 2
460  PRINT "x3 = □" : (-a - SQRd) / 2
```

Opmerkingen :

1. Het programma zoekt een wortel van de vergelijking met behulp van een iteratieve werkwijze, die men de *Newton - Raphson methode* noemt. Deze methode werkt als volgt: men neemt een testwaarde voor x en verbetert ze achtereenvolgens totdat ze de wortel voldoende dicht heeft benaderd. Deze werkwijze wordt uitgevoerd door regels 90 - 150.
2. Indien U wilt volgen op welke manier deze iteratie werkt, dan voegt U :

   ```
   131  PRINT x
   ```

   toe en zult U kunnen zien dat de getallen naar het antwoord toe convergeren.
3. Wanneer het programma een wortel heeft gevonden, wordt de vergelijking hierdoor gedeeld en krijgt men een vierkantsvergelijking die wordt opgelost met behulp van de formule op regels 400 - 460.
4. Indien deze vierkantsvergelijking geen reële wortels heeft, wordt regel 430 toegepast; het programma stopt dan op regel 445.
5. Voor sommige derdegraads vergelijkingen kunnen de onnauwkeurigheden in de rekenkundige bewerkingen zich opstapelen. Het programma kan in deze gevallen de uitlezing "Andere wortels zijn imaginair" geven, hoewel dit onjuist is. Regel 440 waarschuwt de operator ervoor dat deze mogelijkheid niet ondenkbaar is. Het teken van d staat cruciaal in dit probleem, en indien d nadert tot 0 dan kunnen foutjes een rampzalig effect veroorzaken.

6. Indien U bijvoorbeeld, het volgende invoert :

   a = 4, b = −8, c = 5, d = −51, dan moet U als resultaat krijgen : x1 = 3, andere wortels imaginair. Probeer nu a = 4, b = −16, c = −5, d = 51 in te voeren, U krijgt nu : x1 = 3, x2 = 2.6213203, x3 = −1.6213203.

JACKPOT

```
10   LET c = 0
20   DIM a(3)
30   FOR t = 1 TO 3
40   LET r = INT(3 * RND)
50   LET a(t) = r
60   LET i = 5
70   LET j = 10 * t − 5
80   GO SUB 300
90   NEXT t
100  LET c = c − 1
110  IF a(1) = a(2) AND a(2) = a(3) THEN GO TO 700
120  GO SUB 1000
130  INPUT "Stop?" ; b$
140  IF b$ = "s" THEN STOP
150  GO TO 30
300  PRINT INK2 * r , AT i,j ; "■■" ; AT i + 1,j ; "■■"
310  BEEP .1 , 3 * r
320  RETURN
700  LET c = c + 9
720  GO TO 120
1000 IF c > = 0 THEN PRINT AT 18,5 , "U wint □" ; c ; "□ gul-
     dens"
1010 IF c < 0 THEN PRINT AT 18,5 ; "U verliest □" ; −c ; "□ gul-
     dens"
1020 RETURN
```

Opmerkingen :

1. Dit programma toont drie versierde vierkantjes. Indien ze alle drie gelijk zijn, wint U fl 9,–, indien ze niet gelijk zijn, verliest U fl 1.–. U begint te spelen nadat U een willekeurige toets heeft ingedrukt, behalve toets "s". We bevelen de ENTER - toets aan omdat U dan een toets minder moet indrukken. U drukt op "s" om te stoppen.
2. Gemiddeld genomen, haalt U uw onkosten terug eruit ; dit is beter dan wat U in het plaatselijke café krijgt.
3. Let op het gebruik van subroutines om de drie mogelijke types van vierkantjes uit te plotten.
4. Merk ook de speciale schikking van het scherm op; alles wordt netjes in het midden afgebeeld.

FACULTEITSFUNCTIE

De Spectrum laat U een subroutine aanroepen vanuit de subroutine zelf: men noemt dit *recursief* programmeren. Probeer in dit programma eens uit te zoeken hoe alles juist werkt.

```
10   LET f = 1
20   INPUT n
30   LET m = n
40   GO SUB 100
50   PRINT m ; "□ Faculteit is □" ; f
70   STOP
100  IF n < = 1 THEN RETURN
110  LET f = f * n
120  LET n = n - 1
130  GO SUB 100
140  RETURN
```

subroutine

DAGZOEKER

Dit programma voert een datum, d. m. j. in. Hierin is d de dag, m de maand, en j het jaar (vb. 23.7.1066). Het programma berekent welke dag het was, is of zal zijn.

```
10   LET a$ = "033614625035"
20   LET b$ = "ZONMAADINWOEDONVRYZAT"
30   INPUT d
40   PRINT d ; "." :
50   INPUT m
60   PRINT m ; "." ;
70   INPUT j
80   PRINT j ; "□ is □" :
90   LET z = j - 1
100  LET c = INT(z/4) - INT(z/100) + INT(z/400)
110  LET x = j + d + c + VAL a$(m) - 1
120  IF m > 2 AND (j = 4 * INT(j/4) AND j < > 100 * INT(j/100)
     OR j = 400* INT(j/400)) THEN LET x = x + 1
130  LET x = x - 7 * INT(x/7)
140  PRINT b$(3 * x + 1 TO 3 * x + 3)
```

Opmerkingen :

1. Regel 10 slaat een list op van twaalf "correctiegetallen voor de maanden". Dit gebeurt op een compacte manier d.m.v. een string. Regel 100 bepaalt de maand in de list met behulp van een stukje rekenwerk dat in opmerking 4 wordt verduidelijkt.
2. Regel 20 maakt van een gelijksoortig truukje gebruik om de uitprintroutine te vereenvoudigen. Ziet U op welke manier regel 140 de juiste lettergroep selecteert?

3. Regel 100 berekent het aantal schrikkeljaren die zich hebben voorgedaan sinds het jaar nul. Ter herinnering: veelvouden van 4 zijn schrikkeljaren, maar veelvouden van 100 zijn dit niet tenzij het ook veelvouden van 400 zijn.
4. Regel 110 is de kern van de berekening. Men telt het aantal dagen dat verlopen is sinds een bepaalde (willekeurige) referentiedatum, maar om plaats te sparen worden de veelvouden van 7 niet in aanmerking genomen omdat die toch de dag van de week niet beïnvloeden. Dus wordt het aantal jaren (j) vermenigvuldigd met 365; maar 365 = 7 * 52 + 1 dus gebruiken we j in plaats van 365 * j. a$(m) houdt rekening met de bijzonderheden van de lengte van de maanden. Merk het gebruik van VAL op om een karakter, bestaande uit één enkel cijfer, om te zetten in een actueel *getal*. Om het programma te ijken kozen we een datum waarvan we wisten welke dag van de week we toen hadden – 30 9.1981 was een woensdag – en hiervoor konden we de –1 correctie aan het einde uitvoeren. Regel 120 stelt de computatie bij in januari of februari van een schrikkeljaar; indien dit niet werd gedaan, zou alles foutlopen.

BINAIR / DECIMAAL CONVERSIE

Het volgende programma zet een ingevoerd getal om van binaire naar decimale notatie of omgekeerd.

Binair naar decimaal

```
10  INPUT a$
20  PRINT a$ ;
30  LET l = LEN a$
40  LET s = VAL a$(1)
50  FOR j = 2 TO l
60  LET s = 2 * s + VAL a$(j)
70  NEXT j
80  PRINT "□ is □" ; s ; "□ in decimale notatie"
```

Decimaal naar binair

```
10  LET c$ = ""
20  INPUT a
30  PRINT a ;
40  LET d = INT(a/2)
50  LET r = a – 2 * d
60  IF r = 0 THEN LET b$ = "0"
70  IF r = 1 THEN LET b$ = "1"
80  LET c$ = b$ + c$
90  IF d = 0 THEN GO TO 120
100 LET a = d
110 GO TO 40
120 PRINT "□ is □" ; c$ " in binaire notatie"
```

Opmerkingen :

1. De INPUT voor binair/decimaal conversie moet een sequentie zijn van nullen

en énen, zoals 11000101011, voor decimaal/binair conversie moet de invoer een geheel getal zijn, zoals 3427006.

2. Beide programma's werden geschreven als illustratie van de verschillende manieren waarop het conversieprobleem wordt aangepakt. Het is mogelijk om beide programma's qua structuur sterk op elkaar te laten lijken.

3 In de decimaal/binair conversie merken we op dat regels 60 - 70 *niet* kunnen worden vervangen door :

```
60  LET b$ = "r"
```

U kunt het schrijven van twee regels vermijden door CHR$ te gebruiken en er goed op te letten wat U doet, of U kunt schrijven :

```
60  LET b$ = ("1" AND r) + ("0" AND 1 - r)
```

dit werkt om redenen die te maken hebben met de manier waarop de Spectrum logische bewerkingen uitvoert, zie *handboek*.
Het volgende is ook mogelijk :

```
60  LET b$ = "0"
70  IF r = 1 THEN LET b$ = "1"
```

dit werkt perfect en spaart bovendien een beetje ruimte.

BRICKBAT

Er vallen gekleurde stenen uit de lucht. Indien U ze met een "bat" kunt opvangen, verdwijnen ze; indien U ze mist, dan stapelen ze zich op tot zuilen. Wanneer een zuil te hoog wordt is het spel ten einde. Hoeveel stenen kunt U opvangen?
U beweegt de rode bat naar links of naar rechts met de pijltjestoetsen 5 en 8.

```
    BORDER 1 : PAPER 7
10  LET h = 11
20  LET c = 0
30  DIM a(5)
```

```
40  LET p = INT (5 * RND + 1)
50  LET j = 5 * p
60  FOR i = 1 TO 21 - a(p)
70  PRINT AT i,j ; INK 6 * RND ; INVERSE 1 ; "□"
80  BEEP .02,35 - 1.9 * i - j/2
90  PRINT AT i - 1,j ; "□"
100 LET m$ = INKEY$
110 IF m$ = "5" THEN LET h = h - 1
120 IF m$ = "8" THEN LET h = h + 1
130 IF h < 1 THEN LET h = 1
140 IF h > 26 THEN LET h = 26
150 PRINT AT 15,h ; "□" ; INK 2 ; INVERSE 1 ;
    "□□□" ; INVERSE 0 : "□" :
160 IF i = 15 AND ABS(h + 2 - j) < = 1 THEN LET c = c + 1 :
    GO TO 40
170 NEXT i
180 LET a(p) = a(p) + 1
190 IF a(p) = 7 THEN PRINT AT 2,5 : INK 1 ,
    "U onderschepte □" : c ; "□ stenen" : STOP
200 GO TO 40
```

SPIRALEN EN ROZETTEN

```
    BORDER 1 : PAPER 2 : INK 7
10  FOR t = 0 TO 4 STEP .5
15  PLOT 128,88
20  FOR y = 0 TO 720
30  LET x = y * PI/180
40  LET r = 1.5 * x
50  LET a = 128 + 5 * r * COS(x + t)
60  LET b = 88 + 4 * r * SIN(x + t)
70  DRAW a - PEEK 23677,b - PEEK 23678
80  NEXT y
```

Opmerkingen :

1. "PI" in regel 30 is toets "M" in extended mode en niet de letters P en I.
2. Regel 70 gebruikt de systeemvariabele COORDS (zie *handboek*) om de juiste uitgangspositie te berekenen voor het tekenen vanaf de oude PLOT - positie naar de nieuwe. De machine zal een rechte lijn trekken naar de volgende PLOT - positie indien U het commando PLOT p,q vervangt door DRAW p - PEEK 23677,q - PEEK 23678. Men kan dit effect ook op andere manieren bereiken, maar indien U COORD gebruikt, vermijdt U dat fouten worden opgestapeld en dit is met andere methodes niet steeds het geval.

3. Om in plaats van spiralen, rozetten te tekenen, verandert U regel 20 en regel 40 in:

```
20  FOR y=0 TO 360
40  LET r=20+SIN(7*x)
```

4. Experimenteer met de "7" te vervangen door andere waarden zoals 3, 4, 5, 6, 8, 9, 10.

PICASSO

Het programma begint met om een "mode" te verzoeken. Deze is óf "d" van "DRAW", óf "e" van "erase" (uitgommen), óf "f" van "finish" (einde). In het laatste geval wordt het programma beëindigd.
Indien U "d" of "e" invoert, vraagt het programma naar een optie die een getal moet zijn uit het bereik van 1 tot en met 9. Het resultaat van elke optie vindt U hieronder. Denk eraan dat, indien de mode "e" is, "erase" moet worden gesubstitueerd daar waar het woord "draw" wordt gebruikt.

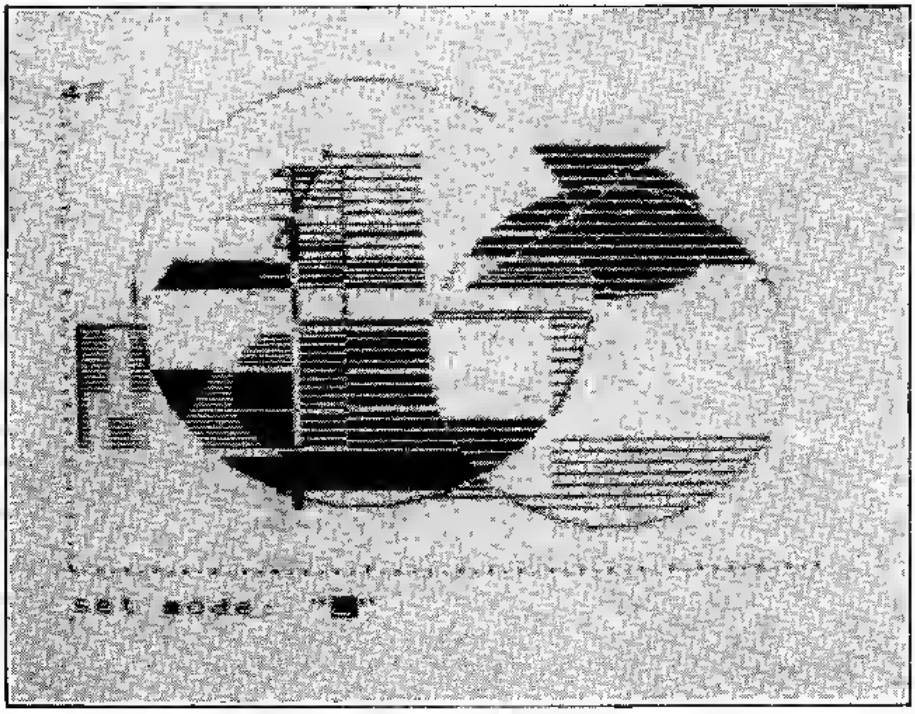
Optie	*Effect*
1	Teken horizontale en vertikale schaalverdelingen als referentie. In elke vijfde kolom en vijfde rij verschijnt een stip, in elke 10de kolom en rij een dubbele stip (streepje) en in elke 100ste een iets langer streepje.
2	Rechthoek. De computer vraagt naar de linker en rechter kolommen en naar de bovenste en onderste rijen waarin de hoeken moeten komen. Dan vraagt de Spectrum of de rechthoek moet worden getekend of dat hij als frame wordt gebruikt (zie optie 7).
3	Cirkel. U moet kolom en rij opgeven voor het middelpunt, en de straal specificeren.
4	Segment. U moet de coördinaten opgeven van de beide hoeken van het cirkelsegment én de grootste afstand meedelen tussen de curve en de rechte lijn.
5	Rechte lijn. U moet de coördinaten van de uiteinden van het lijnstuk opgeven.
6	Gebogen lijn. U moet de coördinaten opgeven van de beide uiteinden van het lijnstuk én de grootste afstand specificeren tussen de gebogen lijn en de denkbeeldige rechte tussen deze twee uiteinden.
7	Schaduwen. De Spectrum vraagt U of de figuur die in het frame ligt moet worden "ingeblokt" of "gearceerd". Merk op dat de gebruikte framerechthoek de enige is die als frame kan dienst doen. In dit verband wordt het woord "in" inclusief gebruikt; m.a.w. indien U een rechthoek tekent en dan optie 7 aanroept dan wordt deze rechthoek geschaduwd, tenzij zich in deze rechthoek een andere figuur bevindt: in dat geval zullen er eigenaardige dingen gebeuren. Indien U "arceringen" koos, wordt U gevraagd naar de afstand tussen elke lijn en óf de arceringen recht of gebogen moeten zijn. Indien ze gebogen moeten zijn, wordt U verzocht de maximum afstand mee te delen tussen de gebogen lijn en het denkbeeldige lijnstuk tussen de beide uiteinden. (Voer eerst optie 2 in.)
8	"Save". Hierdoor wordt de tekening op tape "geSAVEd". U wordt verzocht een naam op te geven.

9 "Load". Hierdoor wordt de tekening van de tape in de Spectrum geladen. U moet de naam van de tekening meedelen.

Opmerking :

Het is niet nodig dat U het programma in zijn geheel gebruikt. De subroutines beginnen op de regels 1000. 2000. 3000 enzovoort, en komen overeen met de opties: 1. 2, 3 enzovoort. Indien U dus met iets eenvoudig wenst te beginnen en slechts rechthoeken en cirkels wilt tekenen, dan heeft U de codes van 4000 en verder niet nodig. Indien U dit doet, moogt U natuurlijk de opties 4 tot en met 9 niet invoeren want dan krijgt U een foutmelding. U moet er ook rekening mee houden dat de routines die beginnen op 4000. 6000 en 7000 alle de routine op 9500 aanroepen.

```
10  INPUT "mode:" : m$
20  IF m$ = "f" THEN STOP
30  IF m$ = "e" THEN OVER 1
40  IF m$ = "d" THEN OVER 0 : INPUT "kleur:" ; c ; INK c
50  INPUT "optie:" ; op
60  GO SUB 1000 * op
70  GO TO 10
```




```
1000  FOR x = 0 TO 255 STEP 5
1010  PLOT x,0
1020  IF x/10 = INT(x/10) THEN PLOT x,1
1030  IF x/100 = INT(x/100) THEN PLOT x,2
1040  NEXT x
1050  FOR y = 0 TO 175 STEP 5
1060  PLOT 0,y
```

```
1070  IF y/10 = INT(y/10) THEN PLOT 1,y
1080  IF y/100 = INT(y/100) THEN PLOT 2,y
1090  NEXT y
1100  RETURN
2000  INPUT "linker kolom van de rechthoek:" ; lc
2010  INPUT "rechter kolom:" : rc
2020  INPUT "onderste rij:" : br
2030  INPUT "bovenste rij:" : tr
2040  INPUT "draw (d) of frame (f):" ; m$
2050  IF m$ = "f" THEN RETURN
2060  PLOT lc,br
2070  DRAW 0,tr - br
2080  DRAW rc - lc,0
2090  DRAW 0,br - tr
2100  DRAW lc - rc,0
2110  RETURN
3000  INPUT "middelpunt ; (kolom dan rij): ", cc,cr
3010  INPUT "straal " : r
3020  CIRCLE cc,cr,r
3030  RETURN
4000  INPUT "een hoek v.h. segment ; (kolom dan rij):" ; c1,r1
4010  INPUT "andere hoek , (kolom dan rij): " , c2,r2
4020  INPUT " max. afstand tussen curve en lijn:" : d
4030  GO SUB 9500
4040  PLOT c1,r1
4050  DRAW c2 - c1,r2 - r1
4060  DRAW c1 - c2,r1 - r2,a
4070  RETURN
5000  INPUT "één uiteinde : (kolom dan rij):" ; c1,r1
5010  INPUT "ander uiteinde ; (kolom dan rij):" ; c2,r2
5020  PLOT c1,r1
5030  DRAW c2 - c1,r2 - r1
5040  RETURN
6000  INPUT "één uiteinde van gebogen lijn ; (kolom dan rij):" ;
      c1,r1
6010  INPUT "ander uiteinde ; (kolom dan rij):" : c2,r2
6020  INPUT "max. afstand tot rechte:" ; d
6030  GO SUB 9500
6040  PLOT c1,r1
6050  DRAW c2 - c1,r2 - r1,a
```

```
6060  RETURN
7000  INPUT "blok in (b) of arceer (h):" ; m$
7010  IF m$ = "b" THEN LET s = 1 : LET a = 0 : GO TO 7070
7020  INPUT "arceerbreedte:" ; s
7030  INPUT "recht (s) of gebogen (c):" ; m$
7040  IF m$ = "s" THEN LET a = 0 : GO TO 7070
7050  INPUT "max. afstand tot rechte:" ; d
7070  FOR r = br TO tr STEP s
7080  FOR c = lc TO rc
7090  IF POINT (c,r) = 1 THEN GO TO 7210
7100  NEXT c
7110  GO TO 7190
7120  LET c1 = c
7130  FOR c = rc TO lc STEP -1
7140  IF POINT (c,r) = 1 THEN GO TO 7160
7150  NEXT c
7160  LET c2 = c
7170  PLOT c1,r
7175  IF m$ = "c" THEN LET r2 = r : LET r1 = r : GO SUB 9500
7180  DRAW c2 - c1,0,a
7190  NEXT r
7200  RETURN
8000  INPUT "voer naam in van te bewaren tekening:" ; p$
8010  SAVE p$ SCREEN$
8020  RETURN
9000  INPUT "voer naam in van te laden tekening:" ; p$
9010  LOAD p$ SCREEN$
9020  RETURN
9500  LET l = 0.5 * SQR((c2 - c1) * (c2 - c1) + (r2 - r1) * (r2 - r1))
9510  LET a = ASN(2 * l * d/(d ↑ 2 + l ↑ 2)) * 2
9520  RETURN
```

LINETTE

Dit is roulette, maar dan op een rechte lijn gespeeld.

```
10  LET w = 100
20  INPUT "Faites vos jeux □" ; b$
30  CLS
40  INPUT "Inzet □" ; a
50  LET w = w - a
60  FOR n = 0 TO 9
```

```
70  PRINT AT 10,2 * n + 6 ; CHR$(48 + n)
80  NEXT n
90  LET r = INT(5 * RND) + 5
100 LET d = INT(10 * RND)
110 LET r = r * 10 + d
120 FOR n = 0 TO r
130 LET x = n - 10 * INT(n/10)
140 PRINT AT 10,2 * x + 6 ; CHRS 143
145 BEEP .05,x
150 PRINT AT 10,2 * x + 6 ; CHR$(48 + x)
160 NEXT n
170 PRINT AT 10,2 * x + 6 ; FLASH 1 ; CHR$(48 + x)
180 IF b$(1) = "e" THEN GO TO 210
190 IF b$(1) = "o" THEN GO TO 220
200 IF VAL b$(1) = d THEN LET w = w + 10 * a
205 GO TO 240
210 IF INT(d/2) = d/2 THEN LET w = w + 2 * a
215 GO TO 240
220 IF INT(d/2) < > d/2 THLN LET w = w + 2 * a
240 PRINT AT 16,9 , w ; "□ chips □"
250 IF w > = 0 THEN GO TO 20
260 PRINT "U bent platzak"
```

Opmerkingen :

1. Na RUN, kunt U ófwel wedden op even of oneven, ófwel op een getal tussen 0 en 9. (Elk woord dat begint met een e wordt gelezen als "even" en alles wat begint met o wordt gelezen als "oneven".)
2. Dan zegt U – door een getal in te voeren – hoeveel U wenst in te zetten. U begint met fl 100. Dit werd in regel 10 vastgelegd.
3. Indien U wint met e of o, krijgt U uw inzet dubbel terug, met een juist getal tien keer.
4. Om ermee op te houden drukt U bij een karakterinvoer op DELETE en dan op STOP, bij een numerieke invoer volstaat een druk op STOP

CIRKELLINETTE

```
10  LET w = 100
30  CLS
40  CIRCLE 123,91,84
50  CIRCLE 123,91,60
60  FOR n = 0 TO 9
70  PRINT AT 10 + 9 * COS(n * PI/5), 15 + 9 * SIN(n * PI/5) ;
    CHR$(48 + n)
```

```
80  NEXT n
82  INPUT "Faites vos jeux:" ; b$
84  INPUT "Hoeveel wilt U inzetten?" ; a
86  LET w = w - a
90  LET r = INT(5 * RND) + 5
100 LET d = INT(10 * RND)
110 LET r = r * 10 + d
120 FOR n = 0 TO r
130 LET x = n - 10 * INT(n/10)
140 PRINT AT 10 + 9 * COS(x * PI/5),15 + 9 * SIN(x * PI/5) ;
    CHR$143
145 BEEP .05,x
150 PRINT AT 10 + 9 * COS(x * PI/5),15 + 9 * SIN(x * PI/5) ;
    CHR$(48 + x)
160 NEXT n
170 PRINT AT 10 + 9 * COS(x * PI/5),15 + 9 * SIN(x * PI/5) ;
    FLASH 1 ; CHR$(48 + x)
180 IF b$(1) = "e" THEN GO TO 210
190 IF b$(1) = "o" THEN GO TO 220
200 IF VAL b$(1) = d THEN LET w = w + 10 * a
205 GO TO 240
210 IF INT(d/2) = d/2 THEN LET w = w + 2 * a
215 GO TO 240
220 IF INT (d/2) < > d/2 THEN LET w = w + 2 * a
240 PRINT AT 10,10 ; w ; "□ chips □□"
250 IF w > = 0 THEN GO TO 82
260 PRINT FLASH 1 ; "Uw geld is op!"
```

Opmerkingen :

1. De instructies voor het gebruik zijn dezelfde als voor Linette.
2. PI is toets M in extended mode.

AUTOMATISCHE MORSESLEUTEL

Dit programma ontvangt via het toetsenbord een bericht en zet het om in Morsecode. Code en bericht verschijnen beide op het scherm; gelijktijdig worden de strepen en punten hoorbaar gemaakt via de luidspreker van de Spectrum.

```
5   CLS
10  INPUT "Berichtje □" : m$
20  FOR i = 1 TO LEN m$
30  LET c = CODE m$(i)
40  IF c < 32 OR c > 32 AND c < 65 OR c > 90 AND c < 97 OR
```

```
     c > 122 THEN GO TO 120
 50  IF c > 96 THEN LET c = c - 32
 60  LET c = c - 64
100  IF c = -32 THEN PAUSE 40: PRINT "□"
110  IF c > 0 THEN PRINT CHR$(c + 64) ; "□" ; : GO SUB 200
120  NEXT i
130  STOP
200  LET k$ = a$(c)
210  LET t = 1
220  IF k$(t) = "1" THEN PRINT INK 6 ; "." ; : BEEP .1,10
230  IF k$(t) = "2" THEN PRINT INK 3 : "-" ; : BEEP .3,10
240  IF k$(t) = "□" OR t = 4 THEN PRINT "□" : RETURN
250  LET t = t + 1 : GO TO 220
500  REM Input Routine
510  DIM a$(26,4)
520  FOR r = 1 TO 26
530  INPUT a$(r)
540  NEXT r
```

Opmerkingen :

1. Om array a$. die de Morsecode opslaat. samen te stellen, moet U beginnen met GO TO 500, vervolgens voert U 26 strings met eentjes en tweetjes in: dit zijn de Morsecodes voor de letters A – Z, waarbij 1 een punt voorstelt en 2 een streep. Hier volgen de strings die U in volgorde moet invoeren :

Letter	String	Letter	String
A	12	N	21
B	2111	O	222
C	2121	P	1221
D	211	Q	2212
E	1	R	121
F	1121	S	111
G	221	T	2
H	1111	U	112
I	11	V	1112
J	1222	W	122
K	212	X	2112
L	1211	Y	2122
M	22	Z	2211

   Let op! De letters mogen niet worden ingevoerd

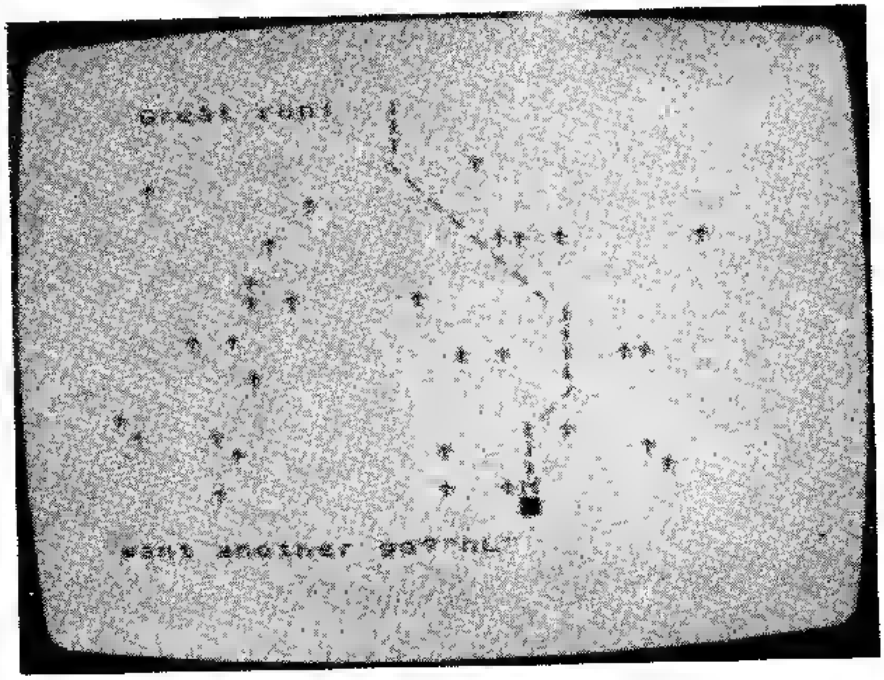
2. Nadat a$ werd samengesteld drukt U op GO TO 5. Druk niet op RUN want dan zou U de variabelen uitwissen die U zopas ijverig hebt ingetypt.
3. Wanneer naar een berichtje wordt gevraagd, typt U er één in. Indien het meer dan 22 karakters bevat, zult U een SCROLL moeten uitvoeren. Het programma

maakt geen onderscheid tussen hoofd- en kleine letters, en alle andere tekens worden genegeerd. Typ bijvoorbeeld "HELLO" in.

4. De computer beeldt nu het berichtje op het scherm af, in letters en in Morse en tegelijkertijd worden de strepen en punten "gebeept".
5. Let op de manier waarop een stringarray wordt gebruikt om de Morsecode in op te slaan. Array's worden dikwijls op deze manier gebruikt: als een "opzoektabel" om het ene codesysteem in het andere om te zetten.
6. Pas het programma aan zodat het letters en cijfers aankan. De Morsecodes voor de getallen 0 - 9 zijn achtereenvolgens:
   -----, ----., ..---, ...--, ...-, ..., -....., --...., ---..., ----..

ST. MORITZ

U moet door een bos skiën en de bomen ontwijken om de skilift te bereiken. U wordt voorgesteld door een "M" en de skilift is een zwarte plek. Het hele parkoers ligt bergafwaarts, zodat U – indien U niets doet – eenvoudig van boven naar beneden komt totdat U op een boom botst. U drukt op "z" om links en op "m" om rechts te draaien.




```
 1  CLS: RANDOMIZE
 2  LET n = 30: LET sc = 14
10  FOR i = 1 TO n
20  LET r = INT(RND * 18) + 3
30  LET c = INT(RND * 32)
40  PRINT AT r,c; "↑"
50  NEXT i
60  PRINT AT 21,20; "■"
```

```
70  PRINT AT 0,sc ; "M"
80  PAUSE 200
90  FOR r = 1 TO 20
100 LET d$ = INKEY$
103 IF d$ = " " THEN PRINT AT r – 1,sc ; "|"
104 IF d$ = "m" THEN PRINT AT r – 1,sc ; "\" : LET sc = sc + 1
105 IF d$ = "z" THEN PRINT AT r – 1,sc ; "/" : LET sc = sc – 1
120 IF SCREEN$(r,sc) = " ↑ " THEN PRINT AT r,sc ; "*crash" :
    GO TO 200
130 PRINT AT r,rc ; "M"
135 PAUSE 2
140 NEXT r
150 IF sc < 22 AND sc > 18 THEN PRINT FLASH 1 ; AT 0,2 ;
    "Bravo!" : GO TO 200
160 PRINT AT 0,3 : "U bent bij de lift aangekomen"
200 INPUT "nog eens?" ; q$
210 IF q$ = "ja" THEN GO TO 1
220 CLS: PRINT AT 10,2 ; "Et maintenant l'après-ski . . ."
```

Wijzigingen en verbeteringen

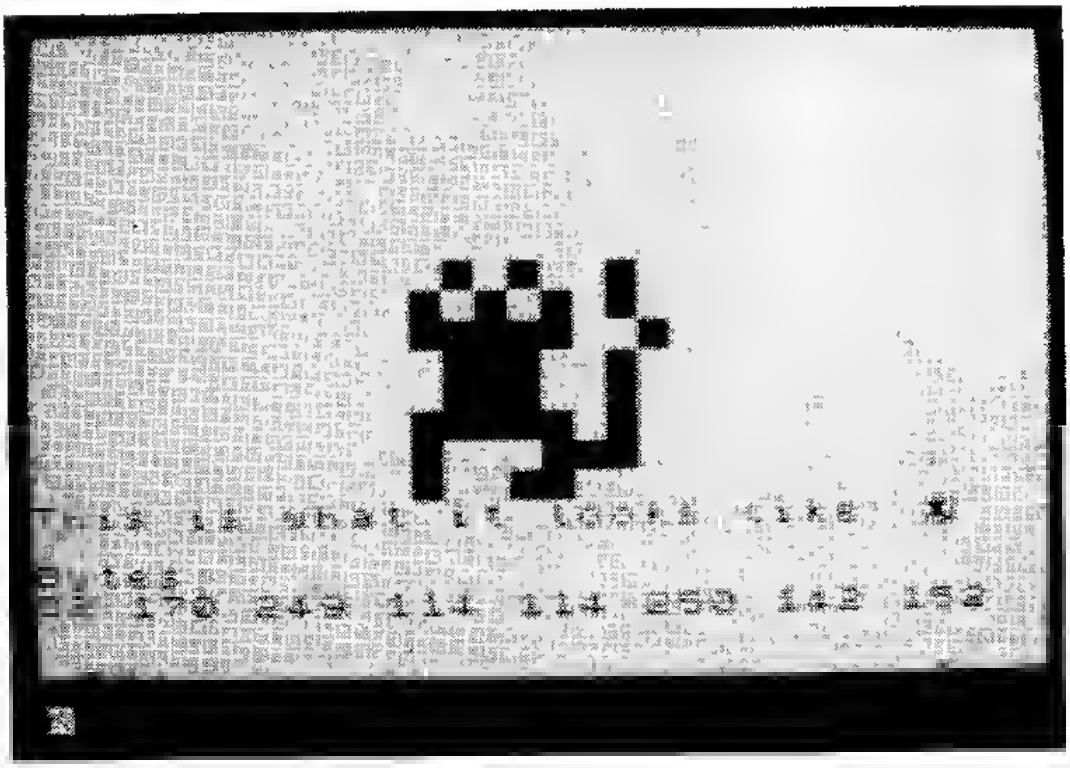
1. U kunt de bomen mooier weergeven. Experimenteer met de karakterbouwer en maak een spar of stel een conifeer voor. Nu U toch bezig bent, wat denkt U ervan om de skiër twee sporen in de sneeuw te laten trekken? Maak ook de skilift mooier.
2. Het aantal bomen werd vastgesteld op 30. Zorg ervoor dat de skiër een moeilijker en een gemakkelijker parkoers kan afleggen.
3. Op regel 135 komt een "PAUSE 2" in de lus voor om een redelijke afdalingssnelheid te geven. Stel een reeks mogelijke pauze-waarden samen en laat de gebruiker een moeilijkheidsgraad (bijvoorbeeld tussen 1 en 5) invoeren zodat hij eigenlijk verschillende pauze-waarden selecteert.

KARAKTERBOUWER

Dit programma maakt het mogelijk een grafisch karakter op een grote schaal op het TV-scherm te tekenen. Daarna wordt het in de computer geladen als het door-de-gebruiker-gedefinieerd-karakter dat overeenkomt met een letter van uw keuze. Het print eveneens de inhouden van de acht bytes uit die de rijen van het karakter vormen. Dit laatste kan nuttig zijn voor eventueel later gebruik in andere programma's.

```
10  DIM k(8,8)
20  FOR i = 8 TO 15
30  PRINT AT i,12 ; "....."
40  NEXT i
50  LET x = 0: LET y = 0
55  GO SUB 500
60  INPUT "Pixel Waarde □" : v
```

```
65   IF v < > 0 AND v < > 1 THEN GO TO 60
70   LET k(x + 1,y + 1) = v
80   PRINT AT x + 8,y + 12 ; INVERSE v ; "□"
90   LET y = y + 1
100  IF y = 8 THEN LET y = 0 : LET x = x + 1
110  IF x = 8 THEN LET x = 0: LET y = 0 : GO TO 200
120  GO TO 55
200  INPUT "Is dit goed?" ; q$
```



```
210  IF q$ = " " THEN GO TO 200
220  IF q$(1) = "y" THEN GO TO 600
230  INPUT "Cursor met pijltjestoetsen verplaatsen" ; j$
235  GO SUB 500
240  IF INKEY$ < > " " THEN GO TO 240
250  IF INKEY$ = " " THEN GO TO 250
260  LET i$ = INKEY$
270  IF i$ = "0" OR i$ = "1" THEN LET k(x + 1,y + 1) = VAL i$ :
     PRINT AT x + 8,y + 12 ; INVERSE VAL i$ ; "□": GO TO 200
275  PRINT AT x + 8,y + 12 : INVERSE k(x + 1,y + 1) ; "□"
280  IF i$ = "5" THEN LET y = y - 1 + (y = 0)
290  IF i$ = "6" THEN LET x = x + 1 - (x = 7)
300  IF i$ = "7" THEN LET x = x - 1 + (x = 0)
```

```
310  IF i$ = "8" THEN LET y = y + 1 - (y = 7)
320  GO SUB 500
330  GO TO 240
500  PRINT AT x + 8, y + 12 ; FLASH 1 ; INK 2 ; "*" : RETURN
600  INPUT "Welke Letter?" : f$
610  FOR n = 1 TO 8
620  LET t = k(n, 1)
630  FOR j = 2 TO 8
640  LET t = 2 * t + k(n, j)
650  NEXT j
660  POKE USR f$ + n - 1, t
670  NEXT n
700  PRINT "This is what it looks like ; ↓" ; CHR$(CODE f$ + 47)
710  PRINT , , "Bytes:"
720  FOR n = 1 TO 8
730  PRINT PEEK (USR f$ + n - 1) ; "□" :
740  NEXT n
```

Opmerkingen :

1. Aanvankelijk vertoont het scherm een array van 8 × 8 stippen en een flitsende cursor; het programma vraagt een Pixelwaarde. U voert 0 of 1 in: indien U 0 invoert, wordt een stip uitgeveegd, indien U 1 invoert wordt de stip een zwart vierkantje. De cursor doorloopt de volledige array automatisch rij na rij. Uw inputs vormen een eerste poging om een bepaald karakter op te bouwen.
2. Dan vraagt het programma "Is dit goed?" Indien U "y" invoert – of iets anders dat met een y begint – dan wordt dit deel van het karakter in de machine geladen (zie opmerking 3). Indien U iets anders invoert dan wordt hetvolgende berichtje uitgeprint: "Gebruik pijltjestoetsen om cursor te verplaatsen". Nu moet U op ENTER duwen en de cursor verschijnt weer. U kunt de cursor sturen door middel van de toetsen 5, 6, 7 en 8 die de cursor in de richting van de pijltjes verplaatsen. Wanneer de cursor op de goede plaats staat, kunt U door een 0 of een 1 via het toetsenbord in te voeren, het overeenkomstig vierkantje uitwissen of inkleuren. Opnieuw verschijnt het berichtje "Is dit goed?" en U kunt – indien nodig – wijzigingen aanbrengen.
3. Als U tevreden bent en U hebt "y" ingetypt dan vraagt de machine U welke letter U met dit karakter wilt associëren. Herinner U dat elk door-de-gebruiker-gedefiniëerd-karakter via het toetsenbord kan worden ingevoerd door middel van een letter (behalve v, w, x, y en z): U moet beslissen welke letter. Indien U bijvoorbeeld "s" kiest dan wordt uw nieuw karakter op de juiste plaats voor "graphics-mode s" opgeslagen. U kunt het ook door middel van de code oproepen als volgt: CHRS(162).
4. De computer geeft ook een listing van de acht bytes die overeenkomen met de rijen van het karakter: dit laat U toe het karakter later opnieuw samen te stellen. U hoeft slechts de getallen op te schrijven en U kunt ze dan achteraf op hun plaats "POKEn" zoals we reeds eerder hebben beschreven.
5. Uw karakter zal van nu af aan totdat U de Spectrum uitschakelt, opgeslagen

zijn als graphics-mode "s". U kunt andere karakters in andere posities stoppen door het programma een nieuwe RUN te geven.

6. U kunt een test uitprinting van uw karakter laten uitvoeren om na te gaan of alles goed is. Indien het nieuwe karakter U niet bevalt drukt U GO TO 200 in. Het karakter wordt niet uitgeprint maar door de cursor heen en weer te bewegen worden de vierkantjes weer zichtbaar en U kunt dan opnieuw wijzigingen aanbrengen.

EEN SCHIETSCHIJF TREFFEN

```
10   LET d = 100
20   LET w = 40 * (.5 - RND)
30   LET v = 1000
40   LET s = 0
50   LET c = 0
100  FOR i = 1 TO 8
110  CIRCLE 127,95,8 * i
120  NEXT i
200  IF c = 11 THEN STOP
205  PRINT AT 20,0 ; "Elevatie = " ;
210  INPUT e : PRINT e
220  PRINT "Deviatie = " :
230  INPUT f: PRINT f
240  LET e = e * PI/180
250  LET f = f * PI/180
300  LET t = d/(v * COS e)
310  LET h = v * t * SIN e - 4.9 * t * t
320  LET k = w * d/100 + d * SIN f
330  LET h0 = 8 * h + 95 + 40 * (.5 - RND)
340  LET k0 = 8 * k + 127 + 40 * (.5 - RND)
350  IF h0 < 0 OR h0 > 175 OR k0 < 0 OR k0 > 225 THEN GO
     TO 500
360  GO SUB 390
370  PRINT AT 20,0 ; "□□□□□□□□□□□□□□□□□□□□", , (16
     spaties)
     "□□□□□□□□□□□□□□□□□□□□" (16 spaties)
375  LET c = c + 1
380  GO TO 200
390  INK 3
400  PLOT k0 - 8,h0 : DRAW 16,0 : PLOT k0,h0 - 8 : DRAW 0,16
405  BEEP .1,5
410  CIRCLE k0,h0,6
415  IF c < 1 THEN GO TO 480
```

```
420  LET q=SQR((k0-127)*(k0-127)+(h0-95)*(h0-95))
430  LET q=INT(q/8)
440  LET q=100-10*q
450  IF q < 30 THEN LET q=0
460  LET s=s+q
470  PRINT AT 0,25 ; c ; TAB 28 ; s
480  INK 0
490  RETURN
500  PRINT AT 0,0 ; "Buiten het scherm"
510  PAUSE 50
520  PRINT AT 0,0 ; "□□□□□□□□□□□□□□□□"          (10 spaties)
530  PRINT AT 20,0 : "□□□□□□□□□□□□□□□□□□□□",,
     (16 spaties)
     "□□□□□□□□□□□□□□□□□□□□"                     (16 spaties)
535  PRINT AT 0,25 ; c
540  LET c=c+1
550  GO TO 200
```

Opmerkingen :

1. Na RUN wordt U gevraagd een elevatie (in graden) in te voeren. De beste waarde hiervoor ligt in de buurt van nul, bijvoorbeeld tussen –2 en 2.
2. Vervolgens moet U de zijdelingse *deviatie* invoeren (om de zijwind te compenseren). Deze moet tussen –10 en + 10 bedragen. NB : Er wordt U niet verteld uit welke richting de wind komt!
3. U krijgt één testschot dat niet meetelt, daarna 10 schoten. Tussen elk schot varieert de wind een beetje.

INDEX

A

AND 30
Array 59
Arraypointer 60
Asverschuiving 78

B

BASIC 9
BEEP 56,60
BIN 51
Bit 87
BORDER 57
BRIGHT 57
Bug 24
Byte 87

C

CAPS SHIFT 17
CHR$ 71
CIRCLE 35
CODE 71
Command 9
Compiler 9
Cursor 10

D

DATA 73
Debugging 24
DELETE 19
DIM 59
DRAW 33
Dry running 44

F

Fibonacci getallen van 23
FLASH 58
FOR NEXT 16

G

GOSUB 52
GOTO 29
Grafiek 45

I

IF . . . THEN 29
INK 56
INKEY$ 53
INPUT 15
Instructie 9
Interpreter 9
Invoer 19

L

LEN 71
LET 14
LOAD 40
Lussen 16, 21
Lussen genestelde 23

M

Machine code 9
Modes 17
Multistatement regel 31

N

NOT 31

O

OR 31
OVER 39

P

PAPER 56
PAUSE 49
PEEK 88
Pixel 32
PLOT 32
POINT 37
POKE 51, 90
PRINT 45
PRINT AT 45
Prompt 25

R

Random number 28
Regelnummer 9
READ 73
RESTORE 74
RETURN 53
RND 28
Runtime error 25

S

SAVE 40
Schaal 76
Schaalfactor 77
SCREEN$ 40
Stapgrootte 24
Statement 9
STEP 24
String 70
Subroutine 52
Substring 71
SYMBOL SHIFT 17
Syntaxis error 25

T

TAB 21
Toetsenbord 17
Toevalsgetallen 28
Top down analyse 85
Tracing 25

U

Uitvoer 20
USR 51

V

Variabelen 13
VAL 71